UM TESTADOR DE ALGORITMOS DISTRIBUÍDOS

Marco Antonio Esteves Galdino

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por :

Prof. Valmir C. Barbosa, Ph.D.
(Presidente)

Eng. George Walter Gerber, M.Sc.

Prof. Aloysio de Castro P. Pedroza, Dr.

Prof. Paulo H. de Aguiar Rodrigues, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
ABRIL DE 1989

GALDINO, MARCO ANTONIO ESTEVES
Um testador de algoritmos distribuídos [Rio de Janeiro] 1989
XII, 173 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1989)
Tese - Universidade Federal do Rio de Janeiro, COPPE
1. Redes de Computadores I. COPPE/UFRJ II. Titulo (série)

A Maria Cláudia

Agradeço à direção do Departamento de Eletrônica do Centro de Pesquisas de Energia Elétrica - CEPEL, na pessoa do Eng. Maurício Moszkowicz, pela oportunidade de realização deste trabalho.

Ao Prof. Valmir C. Barbosa pela orientação deste trabalho.

Ao Prof. Aloysio C. P. Pedroza pela colaboração prestada.

Aos colegas do Departamento de Eletrônica pelo estímulo e apoio na realização desta pesquisa, e em especial a :

. George W. Gerber pela inestimável colaboração prestada, colocando seus conhecimentos à minha disposição e auxiliando todas as etapas deste trabalho.

. Alberto A. Kopiler por assumir outros trabalhos em andamento, possibilitando a realização do presente trabalho.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M. Sc.).

UM TESTADOR DE ALGORITMOS DISTRIBUÍDOS

Marco Antonio Esteves Galdino

Abril de 1989

Orientador : Prof. Valmir Carneiro Barbosa
Programa : Engenharia de Sistemas e Computação

A equipe CDS-Centros De Supervisão do Departamento de Eletrônica do CEPEL (Centro de Pesquisas de Energia Elétrica - ELETROBRAS) desenvolve uma família de sistemas de supervisão e controle aplicados a sistemas elétricos, empregando arquiteturas distribuídas, basicamente redes locais de microcomputadores.

O projeto dos algoritmos distribuídos, e em particular dos protocolos de comunicação, empregados em tais sistemas vinha sendo tradicionalmente realizado de forma pouco estruturada : a partir de uma especificação informal ou semi-formal, baseada em linguagem natural, gráficos, diagramas de estados, etc., obtinha-se uma implementação que era depurada "on-line" no sistema-alvo (sempre distribuído).

O trabalho ora apresentado é um primeiro passo no sentido de construir ferramentas computacionais para o auxílio ao projeto de tais algoritmos. Elas devem permitir um projeto mais sistemático e, se possível, automatizado em algumas etapas.

A sua motivação foi a necessidade de projetar um algoritmo distribuído complexo, que é o gerenciador de bancos de dados replicados para centros de supervisão e controle baseados em arquiteturas distribuídas. Este algoritmo é baseado em um protocolo de difusão confiável.

O trabalho consiste na construção de um testador de algoritmos distribuídos, voltado principalmente para o teste de conformidade de implementações de protocolos de comunicação com as suas especificações. Este testador possibilita a execução de testes manuais e automáticos, sendo que estes últimos são baseados nos conceitos de observadores locais e observador global, que devem ser sintetizados a partir da especificação formal do algoritmo em teste.

O testador proposto suporta implementações reais (adaptadas) e prescinde do sistema-alvo, substituindo-o, de forma simulada em um único processador. Esta abordagem nos confere maior controle sobre a execução dos testes e maior poder de deteção de erros.

Este testador foi implementado num sistema de desenvolvimento MODULA2 para microcomputadores PC-compatíveis. Ele foi exercitado com algoritmos simples, entre eles um protocolo do tipo "sliding window" ponto-a-ponto.

Atualmente alguns algoritmos reais estão sendo adaptados para testes.

Abstract of the Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M. Sc.).

A TESTER FOR DISTRIBUTED ALGORITHMS

Marco Antonio Esteves Galdino

April, 1989

Thesis Supervisor : Prof. Valmir Carneiro Barbosa
Department : Computing and Systems Engineering

The CDS team of the Department of Electronics from CEPEL (Brazil's Utilities Research Center) develops a family of control and supervisory systems for applications in electrical systems. These systems architectures' are distributed, mainly formed by local microcomputer networks.

The project of the distributed algorithms, specially communications protocols, used in these systems, is traditionally made in a non-structured fashion : based in a informal specification (natural language, states diagrams, etc.), an implementation is built. It is tested and debugged in the target-system, which is always distributed.

This is a work towards building tools that help the development of such algorithms, making it more systematic and, if possible, automated in some aspects.

The motivation for this work was the project of a complex distributed algorithm : the replicated database manager for supervisory and control centers based on distributed architectures, which is supported by a reliable broadcast protocol.

A tester for distributed algorithms, wich supports conformance testing of communications protocols, was implemented. This tool allows manual and automatic tests. The automatic tests are based in the concepts of local and global observers, wich are obtained from formal specifications of the algorithms under test.

The proposed tester accepts real implementations (adapted) and substitutes the target-system, simulating it in a single processor. This aproach gives grater control over the test executions and greater error detection power.

This tester was implemented in the MODULA2 development system for PC-like microcomputers. It was validated through the test of simple algorithms, like a sliding window protocol.

Now some real algorithms are being adapted for testing.

ÍNDICE

FIGURAS

CAPÍTULO I

INTRODUÇÃO

CAPÍTULO I

A motivação para este trabalho é o projeto de um protocolo de difusão confiável que será utilizado como suporte para a implementação de um gerenciador de bancos de dados replicados para centros de supervisão e controle baseados em uma arquitetura distribuída [12].

A arquitetura do sistema em questão é uma rede local em barramento, tipo Ethernet. O controle de acesso ao meio é feito pela técnica de múltiplo acesso por detecção de portadora e detecção de colisão (CSMA/CD) e o protocolo de enlace de dados é o HDLC.

A complexidade deste algoritmo implicou na adoção de uma nova abordagem de projeto, diferente da que usávamos até então. Para obter um produto mais confiável e em menor tempo recorremos à construção de uma ferramenta auxiliar para o teste de algoritmos distribuídos.

Não obstante o trabalho ser voltado para protocolos de comunicação, as técnicas e ferramentas aqui descritas são, em princípio, válidas para o projeto de algoritmos distribuídos genéricos, pois os protocolos são apenas um caso particular desta classe de algoritmos.

As metodologias que forneceram o embasamento para este trabalho são as relativas à simulação e ao teste de implementações de protocolos de comunicação.

I.1 - TÉCNICAS BÁSICAS

As técnicas de simulação de protocolos de comunicação são voltadas para a sua validação, ou seja para a verificação de suas propriedades lógicas. As ferramentas de simulação exploram conceitos que são de nosso interesse:

. A simulação implica em alguma forma de execução dos algoritmos, normalmente execução simbólica de várias ocorrências destes.

. As entradas para o sistema simulado são chamadas às primitivas de serviço das ocorrências ("SAPs - Service Access Points").

. A verificação das propriedades lógicas é, muitas vezes, feita por observadores passivos, que monitoram as chamadas às primitivas dos SAPs e o estado interno das ocorrências.

Tais ferramentas proporcionam diversos recursos de interface homem-máquina que facilitam o trabalho do usuário (operador do simulador), tais como : acompanhamento em tempo real da evolução da simulação, geração de listas históricas de eventos, etc.

Podemos distinguir dois modos de operação das simulações: manual, que é realizado interativamente pelo usuário; e automático, onde o simulador, a partir da configuração de parâmetros de simulação, aplica testes e verifica erros automaticamente.

Os testes de conformidade de implementações de protocolos de comunicação visam determinar se as ISTs (Implementações Sob Teste) são consistentes com as suas especificações.

Estes testes são, às vezes, realizados em ambientes remotos, sobre um serviço de comunicação existente, que é o próprio sistema-alvo. Em outros casos eles são realizados em ambientes locais, que substituem o serviço de comunicação. De qualquer forma são conceitualmente semelhantes à simulação para a validação, distinguindo-se pelo fato de executarem implementações reais.

I.2 - ABORDAGEM EMPREGADA

A abordagem que adotamos foi a construção de uma ferramenta auxiliar para o teste de implementações de algoritmos distribuídos. Esta ferramenta emprega os conceitos aqui discutidos. Ela foi implementada no microcomputador PC-compatível, na linguagem MODULA2 e usando um "software" básico já existente, desenvolvido no CEPEL.

O testador controla a execução em tempo real (segundo a estratégia "time-slicing") de diversas ocorrências de implementação de um algoritmo e substitui a rede de intercomunicação, fornecendo um conjunto de serviços para as ISTs.

Os serviços fornecidos pelo testador incluem, além das facilidades para envio e recepção de dados (estímulos às primitivas de seus SAPs), outras para instalação, configuração, etc.

A arquitetura do testador permite, de acordo com a conveniência do usuário, modelar o serviço do nível inferior do protocolo, bem como o comportamento de seu nível superior. Estes modelos são específicos para o algoritmo em teste e devem ser codificados pelo usuário.

As ISTs são bastante próximas de implementações reais, mas devem obedecer a algumas restrições e sofrer um conjunto de adaptações.

É necessário adaptar as ISTs para o acesso aos serviços do testador, e suas restrições principais são : não podem realizar E/S; não podem usar memória dinâmica.

A interface homem-máquina é amigável e funciona através de janelas e menus. Ela fornece ao usuário (operador do teste) facilidades como : disparar e suspender manualmente o teste a qualquer instante; introduzir manualmente entradas para as ISTs - exercitar os SAPs das ISTs; visualizar saídas

geradas pelas ISTs, registro histórico de eventos, etc, que são úteis tanto para testes automáticos quanto manuais.

Nos testes automáticos a detecção de erros é feita por meio de múltiplos observadores, que são específicos para o algoritmo em teste e devem ser codificados pelo usuário.

A fim de exercitar e demonstrar o testador foram realizados diversos testes manuais e automáticos, sendo que para os últimos foram realizadas as sínteses dos observadores correspondentes.

Um protocolo do tipo "sliding window protocol", que é muitas vezes usado para demonstrar e exemplificar técnicas relacionadas ao projeto de protocolos de comunicação, foi implementado com o auxílio do testador. Neste caso os observadores foram obtidos a partir de um modelo formal (Redes de Petri) do protocolo.

O algoritmo MH ("minumum hop-distance"), que obtem as tabelas de roteamento estático, segundo distâncias mínimas, em redes de topologia arbitrária, foi também implementado. Este é um exemplo de algoritmo distribuído que não é voltado à comunicação de dados. Aqui os observadores foram obtidos a partir de assertivas lógicas sobre as variáveis internas das ocorrências do algoritmo.

A experiência indicou que o testador é uma ferramenta bastante útil.

Segundo a avaliação realizada a partir de coletâneas de erros localizados nos testes, o teste manual é eficiente na detecção de erros de programação simples, e o teste automático é capaz de rastrear erros mais complexos.

As experiências permitiram propor diversas extensões do testador, que incrementam a sua capacidade de realização de testes manuais. Já testes automáticos implicam em pesquisar técnicas de obtenção de observadores e de configuração de

testes.

Estamos atualmente adaptando para o testador implmentações de algoritmos reais.

As contribuições acadêmicas que podem ser apontadas neste trabalho são :

. Uma pesquisa bibliográfica acerca das técnicas de projeto de protocolos de comunicação, em particular das técnicas voltadas à validação e ao teste de conformidade.

. A proposta e implementação de uma ferramenta de teste genérica, cuja arquitetura é configurável pelo usuário, e que permite a realização de testes automáticos e manuais. São ainda sugeridas diversas extensões no sentido de aprimorar o testador.

. A realização de exemplos práticos de testes de algoritmos simples.

I.3 - ORGANIZAÇÃO DO TEXTO

O restante do texto é organizado da seguinte forma :

. O capítulo II descreve conceitos e técnicas relacionadas ao projeto de protocolos de comunicação. Ele culmina com a proposta de construção de um testador de implementações de algoritmos distribuídos.

. O capítulo III trata da arquitetura e da implementação do testador proposto.

. O capítulo IV descreve o conjuto de testes utilizado para depuração, demonstração e avaliação do testador.

. O capítulo V contém uma conclusão, sob forma de uma avaliação da capacidade do testador e de propostas de

extensões para o mesmo.

CAPÍTULO II

PROJETO DE PROTOCOLOS DE COMUNICAÇÃO

CAPÍTULO II

A experiência mostra que o projeto dos protocolos de comunicação implica nas seguintes atividades interrelacionadas : especificação, validação, implementação, testes de conformidade e avaliação de desempenho.

. Especificação
Consta de uma descrição do serviço que o protocolo deve fornecer, bem como do detalhamento funcional de sua arquitetura. Os protocolos são normalmente subdivididos em níveis ou camadas e estas descrições são feitas nível a nível.

. Validação
Chamamos de validação de um protocolo à verificação do comportamento de sua especificação em vista da descrição do serviço que este deve fornecer. A validação determina se o protocolo é logicamente consistente, detectando seus erros de modelagem e/ou concepção.

. Implementação
Uma vez completada e validada a especificação ela é usada como guia para a codificação dos programas que irão implementar o protocolo.

. Teste de Conformidade
O teste de conformidade é a verificação do comportamento de uma implementação do protocolo tendo em vista a sua especificação, ou seja, visa detectar erros de implementação.

. Avaliação de Desempenho
A avaliação do desempenho de um protocolo procura levantar parâmetros como taxa de transferência de dados, taxa de ocupação de canais de comunicação, etc.

A base para este trabalho foi uma investigação sobre

técnicas relacionadas às diversas etapas do projeto de protocolos de comunicação. A partir deste estudo foi proposta uma ferramenta para auxiliar ao projeto do protocolo de difusão confiável.

Não existem ainda técnicas padronizadas para abordagem das diversas etapas de projeto. Tais técnicas são objeto de intensa pesquisa, cujo objetivo é a sua padronização e a construção de ferramentas integradas, obtendo o que podemos chamar de estações de trabalho para o auxílio ao projeto de protocolos.

Uma discussão, feita com base na pesquisa bibliográfica, acêrca dos tópicos relativos ao projeto de protocolos que são considerados pertinentes a este trabalho é realizada nos próximos itens. Os tópicos abordados são :

. Especificação;
. Validação por simulação;
. Testes de conformidade.

II.1 - ESPECIFICAÇÃO

O trabalho de especificação de um protocolo pode ser subdividido em duas fases :

. Especificação de serviço
É a primeira fase do projeto. Esta é uma especificação abstrata e voltada para o usuário, ela consiste de um conjunto de sentenças lógicas de alto nível que descrevem o comportamento global do protocolo (ou de determinado nível deste) em vista do serviço que este deve fornecer, por exemplo : "dois nós não podem possuir a permissão de acesso a um determinado recurso simultaneamente", "em todos os nós as mensagens recebidas devem ser passadas ao nível superior na mesma ordem", "um nó não pode solicitar a mútua exclusão de um recurso que ele já possui", etc.

. Especificação funcional
É uma descrição voltada para o sistema. Ela descreve a arquitetura do protocolo, sua divisão funcional, algoritmos, máquinas de estados, etc. Durante o decorrer do projeto normalmente é necessário fazer diversas especificações funcionais, com crescentes níveis de detalhe - O limite deste detalhamento é a própria implementação (abordagem "top-down").

Estas especificações dos protocolos são tradicionalmente realizadas de maneira informal ou semi-formal, por meio de linguagem natural, gráficos, tabelas, diagramas de estados, etc.

O problema da especificação de tais sistemas distribuídos é mais complexo que o dos tradicionais sistemas sequenciais, pois envolve a descrição da sincronização e cooperação de processos e máquinas paralelos, bem como do não-determinismo destes sistemas.

A abordagem tradicional para a especificação é falha. Ela dá margem a ambiguidades e mal-entendidos, dificulta a validação, a implementação e principalmente a construção de ferramentas computacionais para o auxílio às diversas etapas do projeto. Por isso recorre-se às técnicas de especificação formal (FDT's - "Formal Description Techniques").

A técnica formal empregada na especificação de um protocolo condiciona as técnicas utilizadas nas demais atividades do projeto. Existem inúmeras FDTs, algumas delas estão em desenvolvimento em organismos internacionais de normalização (ISO, CCITT) com vistas a se tornarem padrões para a descrição de sistemas distribuídos (embora o principal objetivo destes organismos seja a obtenção de um padrão para a descrição de seus sistemas abertos - "open systems").

A utilização das FDTs apresenta as seguintes vantagens :

. Elimina as ambiguidades das descrições em linguagem natural e fornece uma especificação clara e concisa. Isto é fundamental em muitos casos : sistemas complexos, que exijam o trabalho em equipe; protocolos padrão como X-25, MAP, TOP; etc.

. As diversas FDT's possibilitam, em maior ou menor grau, a construção de ferramentas automáticas para o auxílio às outras etapas do projeto

. O projeto como um todo se torna mais sistemático e o produto final é mais confiável que o obtido pelo processo tradicional.

As principais FDTs são :

. Redes de Petri [9] [24]

. ESTELLE (ISO) - Extended State Transition Language [25]

. LOTOS (ISO) - Language of Temporal Ordering Specification [26]

. SDL (CCITT)-Specification and Description Language [32]

Além destas, outras técnicas menos divulgadas existem : FAPL(IBM), ASYL, ASPEL, PDIL, PASS, etc.

Muitas das FDTs são basicamente técnicas de descrição de modelos estendidos de máquinas de estados finitos (Redes de Petri, ESTELLE, SDL, etc). Outras empregam outros conceitos (LOTOS-Lógica Temporal, etc).

Atualmente o uso das FDTs está se tornando relativamente disseminado. A sua aceitação pela comunidade internacional vem se dando paulatinamente à medida que se realiza o treinamento no seu uso e o desenvolvimento de ferramentas

nelas baseadas para o auxílio aos projetos. Uma vez que estas FDTs ainda estão em desenvolvimento, a maioria das ferramentas atualmente disponíveis suporta apenas um subconjunto destas.

Com a proliferação das FDTs surge a dúvida na escolha da técnica a ser adotada por determinada instituição e/ou para determinado projeto. Existe uma literatura acerca da comparação de vantagens e desvantagens de muitas FDTs com relação a itens como poder de modelagem, ferramentas disponíveis, etc [11] [2] [14] [15]. Entretanto espera-se que em um futuro próximo as FDTs utilizadas convirjam para as que forem outorgadas como padrões pelos organismos internacionais, e que as estações de trabalho integrem os conjuntos de ferramentas disponíveis para cada uma destas FDTs [11].

Algumas FDTs equivalem a linguagens de programação voltadas para sistemas distribuídos, e existem compiladores capazes de traduzir estas descrições para outras linguagens (PASCAL, C, etc.), fornecendo o que é usualmente chamado de especificação executável. Esta é provavelmente a maior vantagem das FDTs, pois tais especificações executáveis são fundamentais nas diversas etapas dos projetos de protocolos [13].

Além dos compiladores, outras ferramentas baseadas em FDTs são: editores sintáticos, interpretadores, simuladores/ depuradores (abordados no próximo item), provadores de propriedades, etc.

II.2 - VALIDAÇÃO POR SIMULAÇÃO

Neste item são discutidas as técnicas de simulação utlizadas para a validação de protocolos de comunicação.

Pode-se distinguir entre duas estratégias de validação :

. Análise de Auto-Consistência
Investiga a consistência interna das diversas partes da especificação - interbloqueios ("deadlocks"), alcançabilidade de estados, etc.

. Análise de consistência com outra especificação
A especificação do protocolo é testada com relação a outra especificação, usualmente mais abstrata. Pode-se por exemplo verificar a primeira especificação funcional com relação à especificação de serviço, ou uma especificação funcional com outra especificação funcional menos detalhada.

O procedimento de validação de um protocolo é iterativo. A tentativa de validação de determinado algoritmo aponta seus erros de concepção, o que induz à revisão de sua especificação e consequente revalidação. Este procedimento produz especificações cada vez mais detalhadas ("top-down").

A justificativa desta abordagem é o fato de que a detecção e correção dos erros de modelagem e/ou concepção é mais fácil, rápida, e apresenta um custo menor, se feita nas primeiras etapas do projeto, uma vez que disponhamos do suporte apropriado.

As falhas que tentamos detetar na validação são decorrentes de conjunções de fatores não previstas na especificação, mas que podem ocorrer eventualmente. Portanto a função do ambiente de simulação é : gerar artificialmente uma variedade de situações possíveis e permitir o rastreamento da evolução do protocolo sob estas a fim de verificar a validade de suas propriedades lógicas.

Uma vez que normalmente não se consegue simular exaustivamente todas as situações possíveis, a simulação eventualmente não localiza todos os erros existentes nos protocolos, mas é capaz de aumentar em determinada quantia, não mensurável, a nossa confiança neles.

A detecção de erros é realizada a partir da verificação de propriedades lógicas do protocolo. As propriedades verificadas podem ser relacionadas à análise de auto-consistência ou à análise de consistência com outra especificação.

A validação por simulação implica em alguma forma de execução das especificações. Um opção consiste na execução simbólica de ocorrências da especificação executável, realizada por um simulador por eventos discretos, paralelamente ao teste de suas propriedades.

Supondo que modelemos o protocolo em Redes de Petri, podemos utilizar ferramentas que executem tais modelos. Estas são às vezes chamadas de jogadores de Redes de Petri.

Com relação ao projeto de protocolos de comunicação a Rede de Petri apresenta limitações relacionadas à dimensão dos problemas envolvidos ("explosão" dos modelos) e aos recursos computacionais necessários para sua análise. Todavia seu uso é disseminado [6] [27], sendo ela voltada para a análise de auto-consistência.

O jogador de Redes de Petri procede da seguinte forma :

. Recebe uma marcação inicial e de uma sequência de fichas de entrada;

. Determina, a partir da marcação corrente, as transições habilitadas;

. Seleciona aleatóriamente (a fim de simular o não-determinismo) para disparo uma transição habilitada;

. Dispara a transição, retirando e inserindo fichas nos lugares associados à mesma;

. Repete o processo enquanto existir transição habilitada.

As transições são consideradas atômicas.

A outra solução possível é a "força bruta". Neste caso o procedimento de execução da Rede de Petri é diferente : todas as transições habilitadas são disparadas e obtém-se assim a árvore de marcação correspondente. Existem ferramentas que usam esta abordagem , isto requer grandes recursos computacionais e é chamado às vezes na literatura de simulação exaustiva.

A análise é feita a partir de propriedades inerentes às Redes de Petri, como alcançabilidade ("reachability"), vitalidade ("liveness"), etc, que devem ser traduzidas e interpretadas como propriedades do sistema modelado, por exemplo ausência de interbloqueios, de violações de exclusão mútua, etc.

A simulação de modelos realizados em outras FDTs adota procedimento análogo (a outra solução possível é tambem a "força bruta").

Para ESTELLE, que é uma FDT que permite construir modelos de uma forma muito mais estruturada e próxima da realidade que as Redes de Petri, a simulação pode ser realizada da seguinte forma :

. A partir da descrição em ESTELLE gera-se diversas ocorrências de especificação executável com as seguintes características : para cada transição são produzidas duas rotinas , uma para avaliação da condição (que é feita a partir de variáveis globais ou locais e de interações com os outros módulos) e outra que realiza a ação associada à transição;

. Este conjunto inerte de rotinas é executado pelo núcleo do simulador por eventos, que é um programa sequencial. Ele implementa as primitivas de comunicação da linguagem ESTELLE.

. O simulador chama todas as rotinas de avaliação de condições para determinar as transições habilitadas;

. Seleciona aleatóriamente uma transição habilitada de cada ocorrência (cada ocorrência é um conjunto de processos concorrentes), simulando assim a o não-determinismo e a execução simultânea das ocorrências;

. Dispara as transições selecionadas chamando sequencialmente , em uma ordem arbitrária, as rotinas associadas às mesmas;

. Repete o processo enquanto houver alguma transição habilitada.

Neste processo de simulação as transições, as ações a elas associadas e as chamadas às primitivas do sistema são consideradas operações atômicas e não consomem tempo.

As descrições dos procedimentos de execução simbólica para Redes de Petri e ESTELLE são didáticas e simplificadas. Na prática é comum que se adote diversas extensões e complicações destes modelos, como :

.Redes de Petri - fichas coloridas , contabilização de tempo virtual de simulação , transições temporizadas , etc.

.ESTELLE - contabilização de tempo virtual de simulação, transições temporizadas, atributos de processos (influem no critério de seleção da transição a ser disparada), criação e destruição dinâmica de processos, etc.

As ferramentas de execução simbólica de FDTs apresentam limitações em termos de número de transições, de ocorrências, etc. Existem ferramentas deste tipo pelo menos para ESTELLE [3] [8] [30] e SDL [4], além das Redes de Petri.

Para a realização destas simulações pode ser necessário

modelar, além do próprio protocolo, o comportamento de :

. Nível inferior do protocolo
Para executar a camada (N) de um protocolo, é necessário que o serviço da camada (N-1) esteja disponível.

Uma vez que as ocorrências em teste podem corresponder a qualquer nível de protocolo, inclusive o nível que interage diretamente com o meio físico de comunicação, deve ser possível simular redes arbitrárias e suas propriedades, tais como atraso em canais de comunicação, perdas de mensagens, etc.

. Nível superior
Corresponde a modelar o comportamento do usuário dos serviços do prolocolo.

Três tópicos relacionados a este tipo de simulação são considerados relevantes :

. O teste das propriedades;
. As entradas para as simulações;
. A interface com o usuário.

Eles são detalhados nos próximos itens.

II.2.1 - TESTE DE PROPRIEDADES - OBSERVADORES

Os itens, denominados pontos de observação, monitorados para a detecção de erros são :

. As sequências de interações realizadas através das interfaces das ocorrências do algoritmo em teste.

. Predicados lógicos sobre variáveis das ocorrências em teste. Os predicados são locais quando dizem respeito a apenas uma ocorrência, e são globais quando são relacionados a várias delas. Exemplos de variáveis

monitoradas são : variáveis de estado, contadores, etc.

Ao ente que tem acesso a estes dados e realiza a verificação do protocolo chamamos observador. Se o observador tiver acesso a todos os pontos de observação sua capacidade de detecção de erros é total, ou seja, ele (ao menos teoricamente) é capaz de detectar qualquer erro. Caso contrário o observador apresenta limitações.

O observador é modelado de forma análoga às ocorrências em teste, ele implementa básicamente uma especificação mais abstrata do protocolo. Normalmente a observação é passiva, ou seja, não interfere no teste, e a sua função é reconhecer se as sequências de interações e estados do sistema são válidos [7] [8].

O observador pode ser especificado na mesma FDT utilizada para modelar o protocolo, assim pode-se tambem gerar a sua especificação executável. Isto torna este tipo de sistema de simulação bastante flexível. Neste caso é necessário atribuir ao observador uma maior prioridade de execução, disparando sempre todas as suas transições habilitadas.

As ações associadas às transições do observador são procedimentos especiais, definidos pelo usuário para cada caso. Por exemplo :

. exteriorização de avisos;
. suspensão da simulação;
. registro na lista de eventos, etc.

II.2.2 - ENTRADAS PARA SIMULAÇÃO

As entradas para as simulações são os conjuntos de estímulos que irão exercitar as ocorrências em teste. São eles :

. Chamadas às primitivas de serviço das ocorrências;

. Outros eventos tais como : perda de mensagens, duplicação de mensagens, falha de nó, etc, que são chamados genéricamente de exceções.
As entradas são conjuntos ordenados de estímulos, onde a cada estímulo é possívelmente associado um determinado instante de aplicação. Estes conjuntos são denominados sequências de teste.

Uma vez que as entidades de protocolo são definidas a partir de interfaces com os níveis superior e inferior, as sequências de chamadas às primitivas de serviço devem ser aplicadas através destas duas interfaces - figura II-1.

As exceções são aplicadas no modelo do nível inferior.

Cada sequência de teste visa exercitar determinado conjunto (sequência) de transições do fluxo de controle do protocolo, e, portanto, tem uma certa cobertura de erros.

A obtenção automática de sequências de teste, a partir de modelos formais de protocolos é possível. Para máquinas de estados finitos são conhecidas diversas técnicas, algumas basedas em heurísticas [27] [10]. SARIKAYA [10] também faz referência à obtenção de sequências de teste para algoritmos escritos em ESTELLE e LOTOS. URAL [29] descreve a obtenção de sequências de teste a partir de ESTELLE.

Algumas das dificuldades enfrentadas por estas técnicas são: transições não observáveis, disparadas espontaneamente através de temporizações internas; eliminação de sequências duplicadas; etc.

Normalmente não é possível testar exaustivamente todas as situações que podem ocorrer, pois estas são em número muito grande ("explosão" das sequências de teste). Uma vez que o número de situações cobertas pela simulação é limitado, somente parte dos erros é detetada.

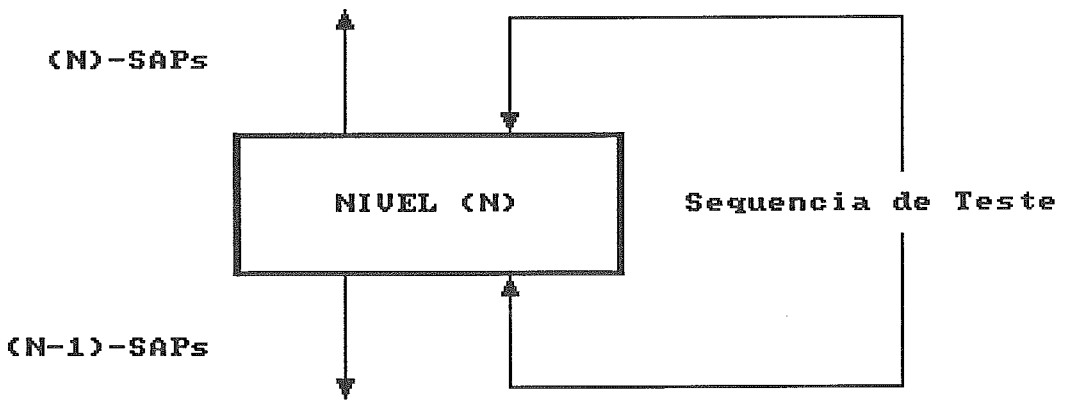


Figura II-1 - Aplicacao de Sequencias de teste

Supondo que a capacidade de observação seja total, a seleção da sequências de teste é o fator mais importante na cobertura de erros da simulação.

Algumas vezes as sequências de teste são geradas "on-line", de forma adaptada à evolução do sistema em teste. Tais sequências, denominadas sequências adaptativas, podem ser vistas como uma árvore de entradas.

A obtenção das entradas para as simulações pode ainda ser feita de forma aleatória. Neste caso usamos geradores aleatórios, com distribuições e parâmetros predefinidos, para gerar estes estímulos. O controle sobre a simulação é feito através destes parâmetros, por exemplo : podemos variar taxas de perdas de mensagens; variar a probabilidade de aparecimento de determinadas sequências de mensagens, etc.

No caso de teste aleatório não podemos definir a cobertura de erros alcançada, entretanto a experiência prova que este tipo de teste é bastante útil em alguns casos [6] [7].

II.2.3 - INTERFACE COM O USUÁRIO

Para serem ferramentas atraentes e de fácil uso para fins de validação, os sistemas de simulação apresentam muitas vezes interfaces homem/máquina confortáveis e poderosas.

Abaixo é apresentada uma lista de algumas das facilidades fornecidas.

. Acompanhamento
O usuário acompanha em tempo real no vídeo a evolução da simulação. Ele visualiza as mensagens trocadas pelas ocorrências do protocolo, os valores de suas variáveis internas, os estímulos aplicados, etc

. Aplicação das sequências de teste

A aplicação de sequências de teste pode ser realizada manualmente pelo usuário e/ou a partir de arquivos préviamente gerados [3].

. Listas de eventos

A forma mais comum de lista de eventos é constituída pelo registro da sequência de interações que ocorrem no decorrer da simulação, o que é usualmente chamado de "trace" na literatura.

Algumas vezes outras informações são incluídas nas listas de eventos, tais como : registro de transições de estados das entidades de protocolo, determinadas condições nas variáveis internas das ocorrências em teste, etc. As listas de eventos assim obtidas são, por vezes, chamadas de históricos de execução [31].

A geração de listas de eventos é muitas vezes realizada pelos observadores. Uma vez detectada alguma situação relevante o observador gera uma mensagem descritiva, que é inserida na lista de eventos. Esta pode ser visualizada "on-line" no decorrer da simulação, ou ainda ser armazenada em arquivos para análise posterior.

A pós-análise pode ser manual ou automática, com o uso de verificadores de listas de eventos ("trace checkers") para a detecção de erros.

O usuário pode configurar os eventos que deseja incluir nesta lista, filtrando o que julgar desnecessario para a análise, a fim de minimizar o volume de informações recebidas.

Pode-se adicionar a facilidade de permitir a qualquer ocorrência gerar uma mensagem arbitrária e incluí-la na lista de eventos.

. Estabelecimento Interativo de "Breakpoints"

O usuário pode especificar interativamente através da interface homem máquina determinados estados do sistema em que deve ser interrompida a simulação. Estes estados são definidos a partir dos pontos de observação [3].

Simuladores que apresentam estas facilidades podem ser utilizados de dois modos [7] :

. Guiado ou Interativo
Neste modo o usuário interage passo a passo na evolução do sistema. Ele introduz as entradas para o sistema, estabelece "breakpoints", consulta e analisa o estado do sistema a cada iteração,etc. O usuário guia manualmente o sistema para determinados estados a fim de verificar se ocorrem erros.

. Automático
O modo automático pode ser aleatório ou por sequência de teste. No primeiro caso a simulação é configurada a partir da especificação dos parâmetros relacionados à geração de entradas aleatórias, no outro a partir da especificação das sequências de teste que devem ser aplicadas.
No modo automático, a partir do disparo da simulação o sistema evolui espontâneamente.
Se houver possibilidade de configurar "breakpoints" nas condições de erro o usuário pode analisar o estado do sistema nessas ocasiões, caso contrário a análise tem de ser totalmente baseada nas listas de eventos. De qualquer forma é conveniente que os observadores registrem nas listas de eventos os erros encontrados.
Uma vez que seja detectado automáticamente algum erro, o usuário pode, a partir da lista de eventos, reproduzir a situação usando o modo guiado.

A figura II-2 apresenta o diagrama em blocos simplificado de um simulador.

II.3 - TESTES DE CONFORMIDADE

O que normalmente chamamos de teste de conformidade é o teste de uma implementação de protocolo para verificar se esta atende aos requisistos para interligação em uma rede existente. Muitas vezes estas redes seguem padrões

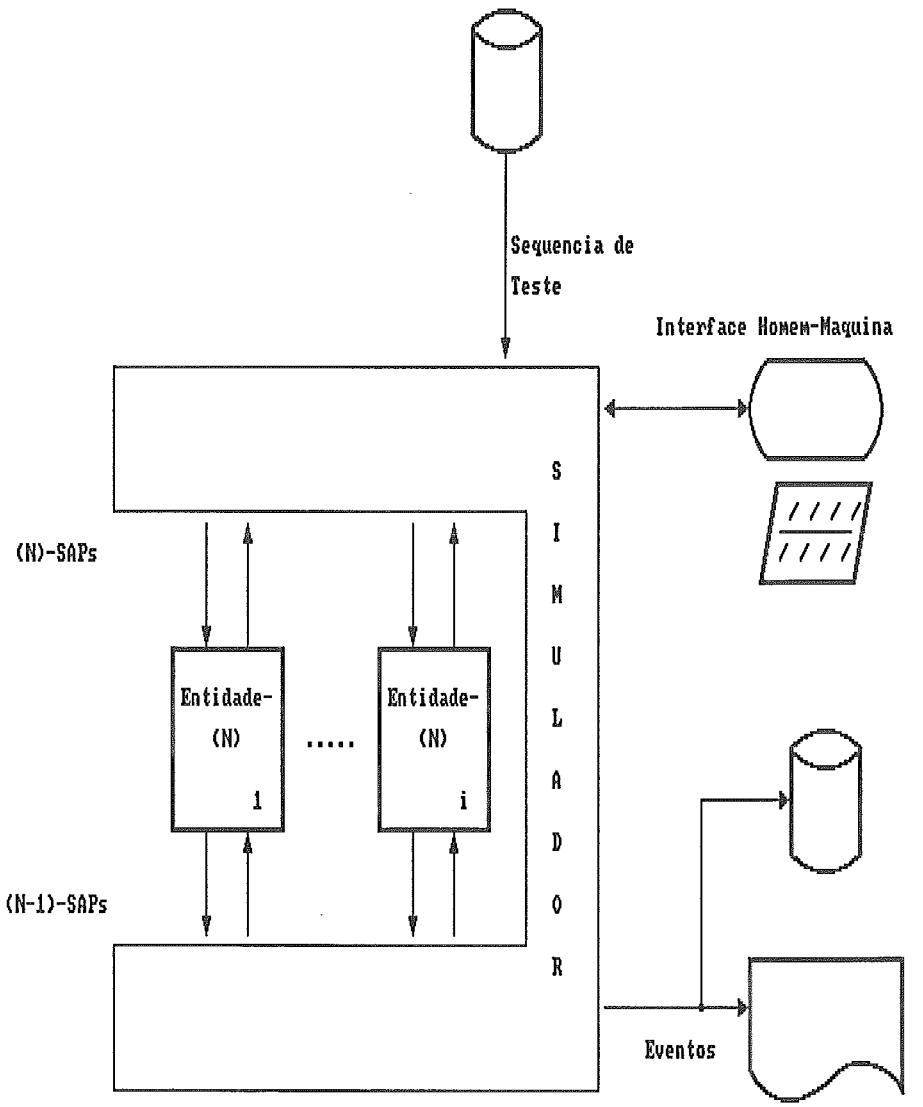


Figura II-2 - Diagrama Simplificado de um Simulador

internacionais ("open systems"), que também fornecem as metodologias de teste.

Tais testes são feitos por centros de certificação de conformidade.

Por extensão chamamos de teste de conformidade a qualquer teste de implementação de um protocolo, que visa determinar se este segue a sua especificação.

A justificativa para a realização destes testes é idêntica à da validação : é mais barato e fácil detetar e corrigir os erros de implementação em uma fase preliminar do projeto.

O método de teste habitual é denominado de teste remoto ou distribuído, cuja arquitetura básica é esquematizada na figura II-3 [10] [23] [27]. O teste do algoritmo é realizado através da execução no próprio sistema-alvo, através de um testador inserido no programa que implementa o protocolo. Suas principais características são :

. A implementação sob teste (IST) é vista como uma caixa-preta;

. A IST e o UT ("upper tester") estão localizados em um nó remoto. O "upper tester" estimula a interface de nível superior da IST;

. O LT ("lower tester") está em um nó local. Ele exercita a interface de nível inferior da IST.

. O serviço do nível inferior disponível ao LT é um serviço "expandido". Ele permite a geração controlada de situações de exceção;

O UT e o LT são os responsáveis pela aplicação das sequências de teste, e, portanto, pelo controle da evolução do teste. Uma vez que eles estão localizados em

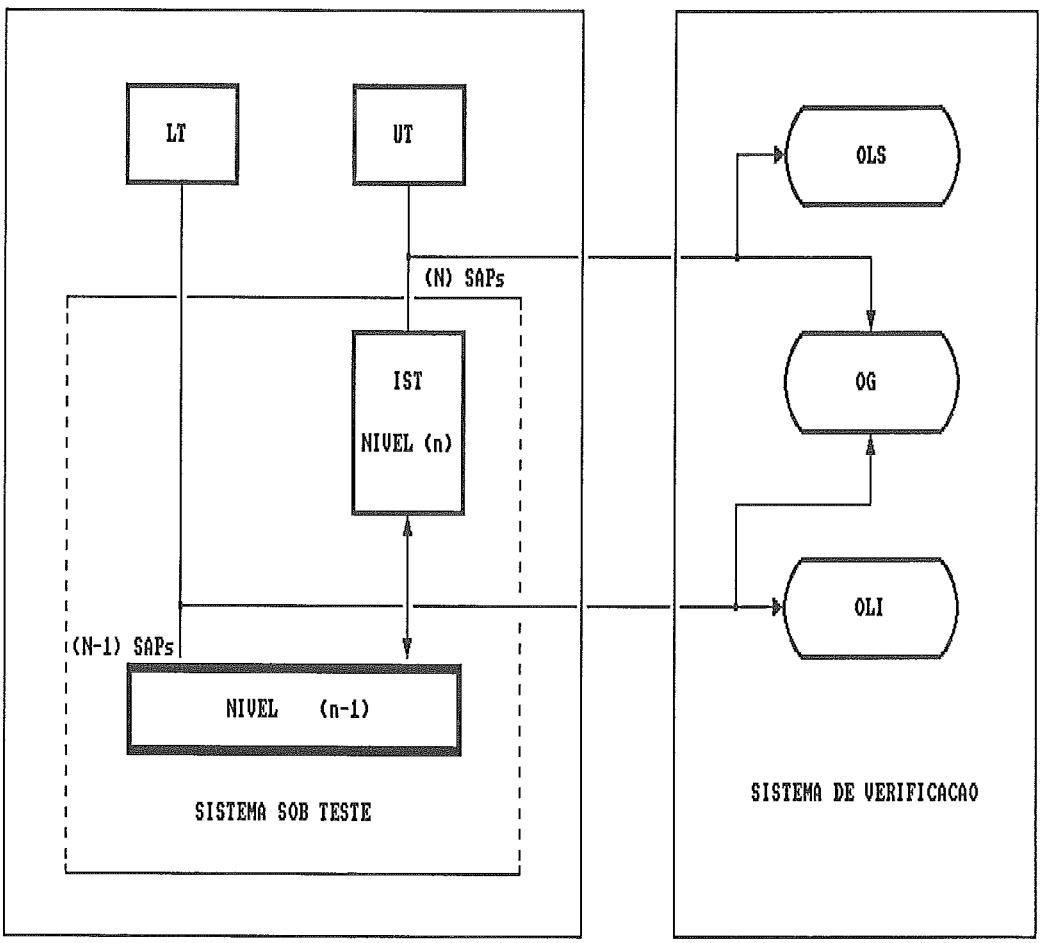


Figura II-3 - Arquitetura de Teste Remoto

nós distintos, é necessária a coordenação de suas atividades. Isto é feito muitas vezes através de protocolos específicos, que apresentam grau de complexidade proporcional à capacidade de controle que fornecem, ou de um canal de comunicação dedicado.

A observação das sequencias de interações é muitas vezes realizada por observadores passivos, que monitoram duas interfaces : a interface do LT com o provedor do serviço do nível inferior e a interface do UT com a IST. Estes observadores são conceitualmente idênticos ao descrito no item II.2.1. Podemos identificar três observadores distintos :

. OLI - Observador Local Inferior
Testa, através da sequência de interações correspondente, o serviço da interface de nível inferior da IST, bem como outras propriedades locais a ela associadas. A presença do provedor do serviço do nível inferior entre o LT e a IST introduz alguma incerteza.

. OLS - Observador Local Superior
Testa o serviço da interface de nível superior da IST.

. OG - Observador Global
O OG testa condições relativas às interações dos dois níveis.

Existem técnicas que indicam como obter e utilizar os observadores passivos de forma a alcançar uma boa cobertura de erros [23].

Diversas variações desta arquitetura são possíveis. A elas correspondem diferentes capacidades de detecção de erros. As variações incluem diversas configurações de observadores e a existência ou não do UT [23].
Outra opção para o teste de conformidade é a realização do mesmo em um ambiente local, com o uso de um testador fora do contexto da rede [27]. Este tipo de ferramenta é

conceitualmente análogo aos simuladores para a validação, entretanto deve executar implementações (ISTs) ao invés de modelos.

Esta abordagem nos confere, em princípio, maior poder de deteção de erros do que a de teste distribuído : seu poder de deteção é total e os pontos de observação são fácilmente acessíveis (o que, na maior parte dos casos, não é verdade para o teste distribuído). Ela não é utilizada nos testes de conformidade para "open systems", pois esbarra na seguinte restrição : Uma vez que a IST é uma implementação, é necessário que o teste seja realizado numa máquina igual à máquina-alvo, e num ambiente análogo. Como as as redes são muito heterogênas, seria necessário refazer o testador para cada teste, o que é pouco prático.

Todavia, no caso de famílias de redes proprietárias, onde o hardware e o ambiente são homogêneos, esta estratégia pode ser válida.

II.4 - TRABALHO PROPOSTO

Não temos no CEPEL um histórico de trabalhos de construção de ferramentas de infra-estrutura para o auxílio ao projeto de algoritmos distribuídos.

Nossa equipe tem realizado o projeto de protocolos de comunicação de forma tradicional: a partir de uma especificação informal partimos para uma implementação, que é testada "on-line" no sistema-alvo.

Nos defrontamos agora com a construção de um algoritmo complexo, que requer uma técnica de projeto mais apurada. Considerando que restrições de prazo e de recursos humanos não nos permitem construir o conjunto de ferramentas necessário para a realização de todas as etapas de forma ideal, partimos para a construção de uma ferramenta de auxílio ao teste de implementações.

O trabalho proposto é a construção de um testador inspirado nos conceitos e técnicas estudadas, relacionadas à validação por simulação e ao teste de conformidade de protocolos.

O testador proposto constitui um ambiente local para o teste de implementações de algoritmos distribuídos, análogo ao discutido ao fim do item anterior (item II.3). Em nosso caso esta é uma alternativa atraente pois trabalhamos com redes homogêneas de microcomputadores baseados em microprocessadores da família INTEL 8086/88.

Seus principais requisitos são listados abaixo.

. O testador deve ser implementado em uma máquina da família 8086/88, assim podemos usar nossas implementações reais (convenientemente adaptadas) como ISTs.

. O testador suportar diversas ISTs simultâneamente, controlando a sua execução em tempo real;

. Deve ser possível modelar o serviço do nível inferior do protocolo em teste, bem como de seu nível superior;

. Para o teste automático deve ser possivel a utilização de múltiplos observadores (OG, OLI, OLS) de acordo com a conveniência do usuário;

. O testador deve fornecer ao usuário o maior subconjunto possível dos recursos de interface homem-máquina discutidos no item II.2.3.

As justificativas desta abordagem são :

. O testador proposto é auto-contido, ou seja, não requer o uso de FDTs e o suporte de outras ferramentas (compiladores para FDTs, simulador, etc.), o que é essencial, pois estas estão, a curto prazo, fora do nosso alcance devido às restrições já levantadas;

. A diponibilidade de outras ferramentas incrementa o suporte ao projeto de algoritmos distribuídos, mas não invalida o uso do testador proposto;

. Podemos, através do testador, detectar tanto os erros de implementação quanto os de concepção, cobrindo assim simultaneamente o teste e a validação de algoritmos (muito embora saibamos que isto não é o ideal, pois a experiência prova que a implementação só deve ser realizada após a validação do algoritmo).

CAPÍTULO III

CONSTRUÇÃO DO TESTADOR

CAPÍTULO III

Neste capítulo é feita uma descrição da técnica de construção do testador proposto, que atende a todos os requisistos enumerados no item II.4.

O testador foi implementado no sistema de desenvolvimento MODULA2 para microcomputadores PC-compatíveis. Este sistema é produzido pela LOGITECH SA e dela obtemos atualizações periódicas. [16] [17] [18] [19]

O "software" básico utilizado na implementação do testador é voltado para aplicações em tempo real, e foi desenvolvido, neste mesmo sistema de desenvolvimento, para projetos anteriores. Ele fornece ao usuário recursos para a programação concorrente, para a implementação de interfaces homem-máquina através de janelas de vídeo, para o registro de eventos, etc. [20]

No próximo item é feita uma descrição da arquitetura do testador e no subsequente são abordados detalhes de sua implementação.

III.1 - ARQUITETURA DO TESTADOR

Esta descrição é baseada no diagrama em blocos apresentado na figura III-1. Os blocos que compõem o testador são os seguintes :

. NÚCLEO - NU;
. GERENCIADOR DE TESTE - GT;
. INTERFACE HOMEM-MÁQUINA - IHM;
. REGISTRADOR DE EVENTOS - RE.

Estes blocos são detalhados a seguir.

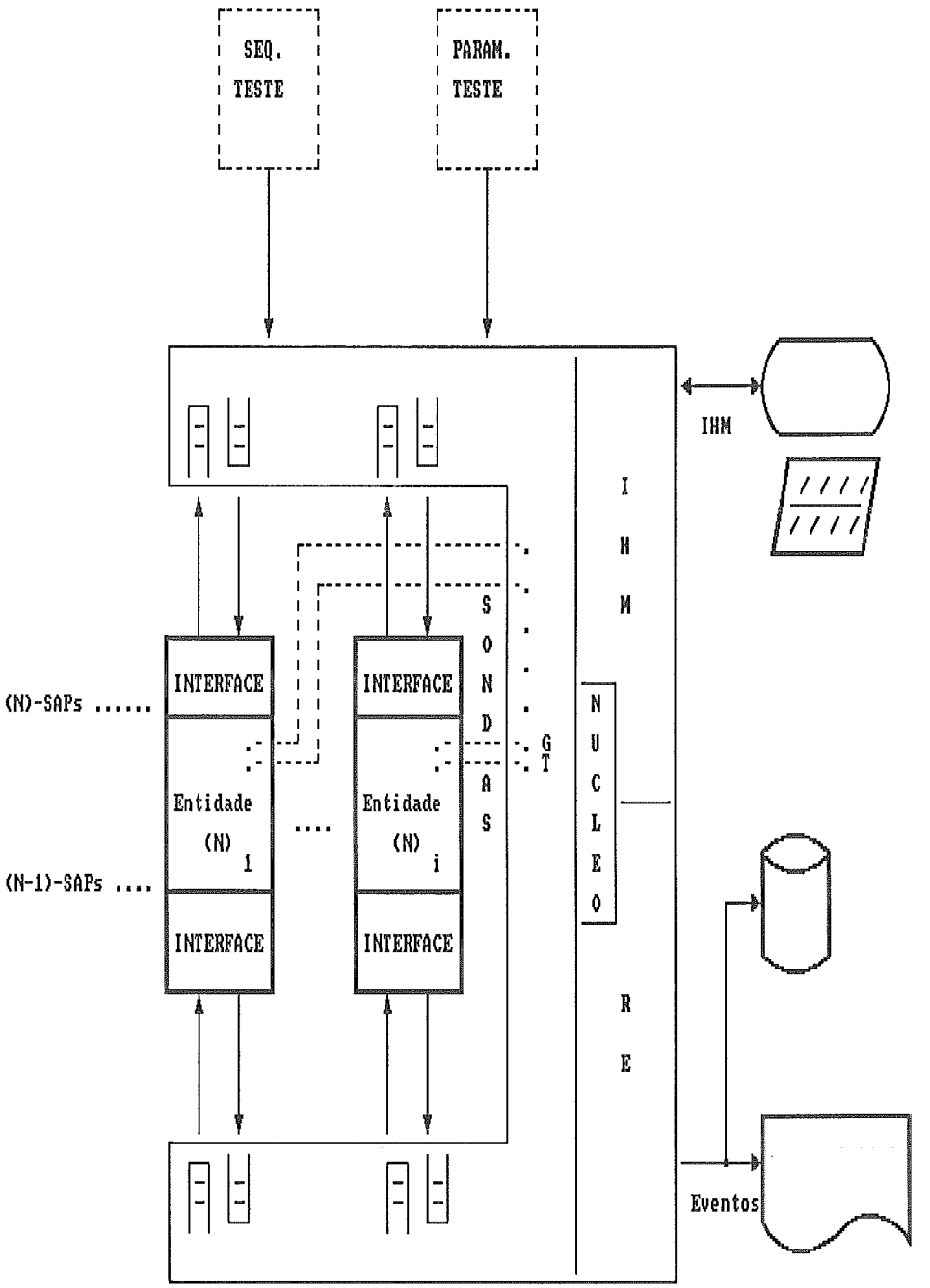


Figura III-1 - Testador Proposto

III.1.1 - NÚCLEO - NU

O sistema em teste - figura III-2 - é composto por um conjunto de subsistemas multitarefa. Um deles é o próprio testador , os outros são as ISTs. Cada IST corresponde, por exemplo, a uma ocorrência do algoritmo distribuído em teste.

O NU cria o ambiente para a execução destes subsistemas. Suas atribuições são :

. Coordenar a execução em tempo real de todos os subsistemas multitarefa que compõem o sistema em teste;

. Prover o conjunto de serviços que será utilizado pelas ISTs;

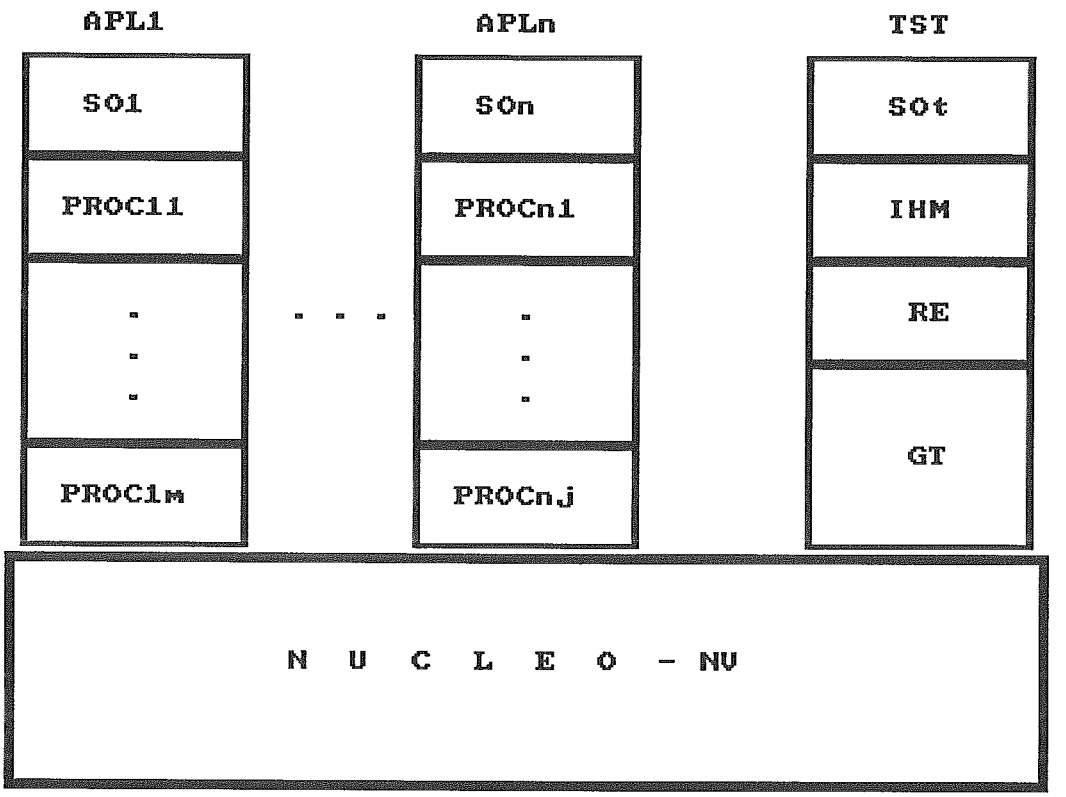
. Prover o conjunto de serviços que será utilizado pelo subsistema testador;

O subsistema testador é composto pelo "software" básico MODULA2 e por um conjunto de rotinas e processos que implementam os blocos da figura III-1 (exceto o NU).

As ISTs são implementações reais, com o mínimo indispensável de adaptações. Elas são auto-contidas e são constituídas por conjuntos de processos concorrentes, sendo que o tratamento da concorrência é realizado internamente. Cada uma delas pode possuir a sua própria estratégia interna de escalonamento, os seus próprios mecanismos de comunicação interprocessos, etc.

Considera-se portanto que o "software" básico das ISTs é simples e modular, semelhante ao do testador. Os recursos disponíveis para as ISTs são : o processador (UCP); e os serviços do testador. Assumir isto é, em geral, razoável para os algoritmos distribuídos em questão.

O subsistema testador é considerado principal e tem a maior



APL1..n => SUBSISTEMAS DE APLICACAO (ESTs)
SO1..n => SISTEMAS OPERACIONAIS DAS APL1..n
PROC11..1M => M PROCESSOS DA APL1
PROCn1..nj => j PROCESSOS DA APLn

TST => SUBSISTEMA TESTADOR
SOt => SISTEMA OPERACIONAL DO TESTADOR
IHM, RE, GT => PROCESSOS DO TESTADOR

Figura III-2 - Nucleo (NU) do Testador

prioridade para a execução. Ele retém o processador indefinidamente, e a execução das ISTs só é realizada sob seu comando.

As varreduras emulam a execução simultânea das ISTs na rede real. Nelas as ISTs são escalonadas sequencialmente segundo uma ordem aleatória. Isto visa evitar a que o testador introduza determinismos no comportamento do sistema em teste.

A estratégia de escalonamento é a fatia-de-tempo ("time-slicing"). Cada IST recebe, por varredura, uma fatia prefixada.

A cada varredura, o contador de tempo virtual de simulação, que registra o tempo decorrido a partir do início desta, é incrementado do valor da fatia.

As varreduras são consideradas atômicas, ou seja, não são interruptíveis, pois neste caso as diversas ISTs estariam em tempos virtuais de simulação diferentes - o que não faz sentido. Após o término de cada varredura o processador é retornado para o testador.

A cada IST estão associadas 4 filas : entrada e saída do nível superior e entrada e saída do nível inferior. Por meio delas é feito o envio e a recepção de dados.

Os elementos destas filas, denominados UDs (Unidades de Dados), contêm, além das informações de controle para o uso do testador, os dados que circulam nas interfaces das ISTs. Os dados contidos em uma UD expressam a invocação de uma primitiva de um SAP ("Service Access Point"), acompanhada dos parâmetros necessários.

A cada IST devem ser atribuídos endereços para as interfaces de nível inferior e superior. O roteamento das UDs para seus destinos é feito segundo estes endereços, que devem ser conhecidos pelos emitentes.

O acesso do testador às variáveis internas das ISTs é feito através de um mecanismo de sondas.

A cada IST está associada uma palavra de estado, que permite configurar, através da IHM, opções como : habilitar e desbilitar a tansmissão de UDs, etc.

Os principais serviços fornecidos pelo NU às ISTs permitem:

. O envio e a recepção de UDs;
. A configuração de sondas;
. A configuração de endereços das interfaces dos dois níveis.

III.1.2 - GERENCIADOR DE TESTE - GT

O diagrama em blocos do GT é apresentado na figura III-3. Cada um dos blocos constituintes do GT será detalhado nos próximos subitens. Uma descrição geral da operação do GT é encontrada no subitem III.1.2.9 - FUNCIONAMENTO DO GT.

Além do conjunto de filas associadas às ISTs, o GT possui um conjunto de filas internas (q1 a q12), que facilitam o fluxo de dados do teste, sob forma de UDs, no seu interior. Elas permitem o fácil acesso de todos os blocos às UDs que lhe são pertinentes.

Com isto ganhamos em modularidade, ou seja, em independência entre os blocos, e em facilidade de sua contrução - em particular dos observadores.

Nos próximos itens serão descritos os blocos constituintes do GT. Estas descrições compreendem os algoritmos de cada bloco, bem como sua filosofia de operação e utilização.

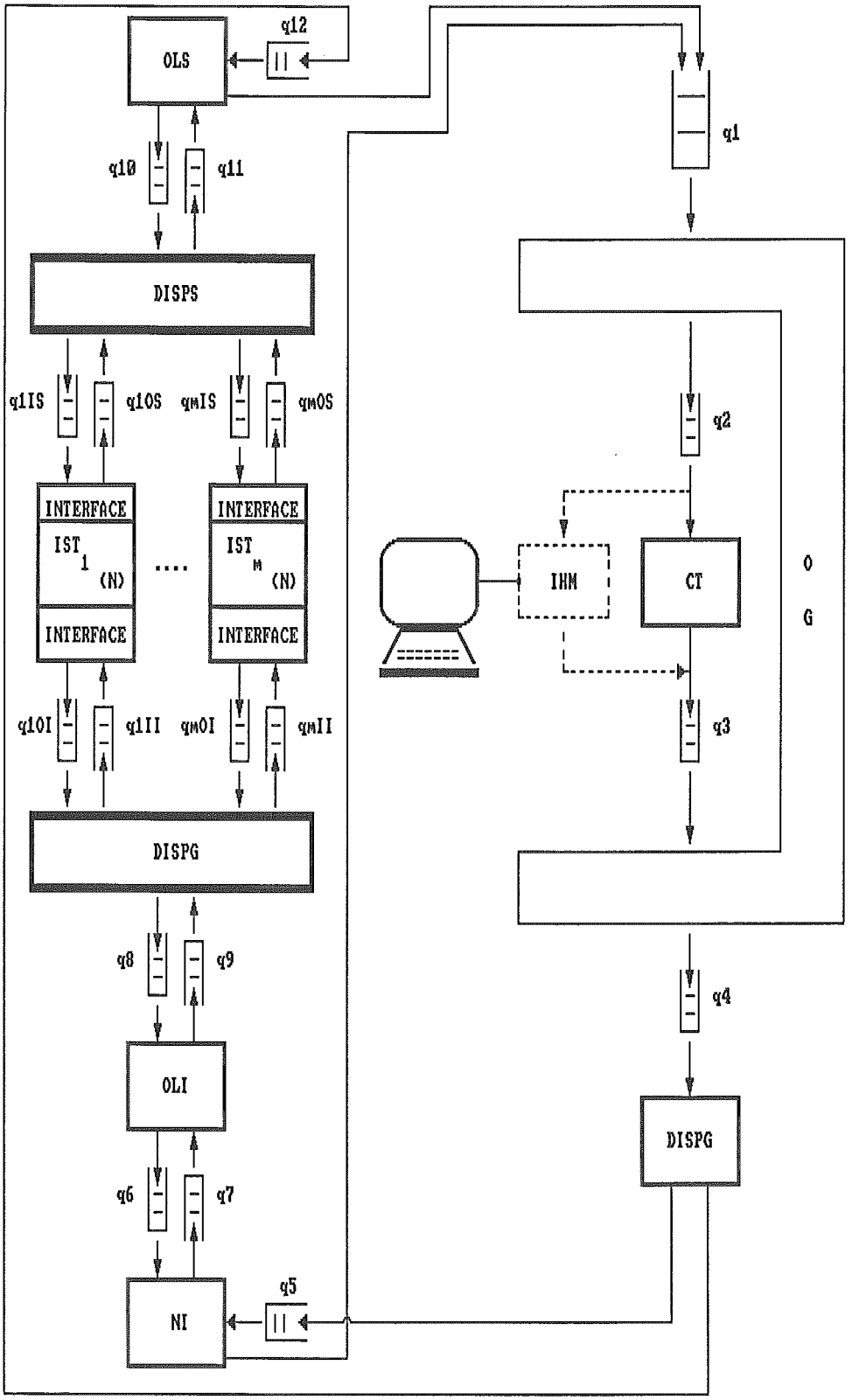


Figura III-3 - Gerenciador de Teste (GT)

III.1.2.1 - DESPACHANTE SUPERIOR - DISPS

O DISPS é simplesmente um roteador das UDs do nível superior. Ele retira as UDs das filas de saída das interfaces de nível superior das ISTs (q1OS a qmOS) e as insere na fila concentradora q11, onde elas se tornam disponíveis para o bloco seguinte do GT (o OLS). Ele também esvazia a fila q10, gerada por este outro bloco, e distribui as UDs ali contidas para os seus destinos na filas de entrada superiores (q1IS a qmIS) das ISTs.

O DISPS realiza ainda o registro automático de eventos de transmissões e recepções de UDs através das interfaces superiores das ISTs ("tracing").

A operação do DISPS é controlada pelas palavras de status das ISTs. Seu algoritmo é apresentado abaixo.

```
PROCEDURE SUPDISPATCHER;
/* Despachante superior */
BEGIN;
   WHILE ( EXISTE ELEMENTO EM Q10 )
   DO BEGIN;
      RETIRA ELEMENTO DE Q10;
      FOR ( TODAS AS ISTs )
      DO BEGIN;
         IF ( ENDEREÇO DESTINO DA UD ESTÁ CONTIDO NO
            CONJUNTO DE ENDEREÇOS DA PORTA SUPERIOR DA IST)
         THEN BEGIN;
            IF ( RECEPÇÃO ATRAVÉS DO NÍVEL
                        SUPERIOR DA IST ESTÁ DESABILITADA )
            THEN REGISTRA EVENTO DE FALHA NA TRANSMISSÃO;
            ELSE BEGIN;
               INSERE COPIA DO ELEMENTO NA FILA
                              DE ENTRADA SUPERIOR DA IST;
               IF ( REGISTRO DE RECEPÇÃO ATRAVÉS
                        DO NÍVEL SUPERIOR ESTÁ HABILITADO )
               THEN REGISTRA EVENTO DE RECEPÇÃO
```

```
                                        SUPERIOR COM SUCESSO;
            END;
         END;
      END;
      DEVOLVE O ELEMENTO AO SACO;
   END;
   FOR ( TODAS AS ISTs )
   DO BEGIN;
      WHILE ( EXISTE ELEMENTO NA FILA DE
                                  SAÍDA SUPERIOR DA IST )
      DO BEGIN;
         RETIRA ELEMENTO DA FILA;
         IF ( A TRANSMISSÃO DE NÍVEL SUPERIOR DA
                                     IST ESTÁ DESABILITADA )
         THEN BEGIN;
            REGISTRA EVENTO DE FALHA NA TRANSMISSÃO;
            DEVOLVE O ELEMENTO AO SACO;
         END;
         ELSE BEGIN;
            INSERE O ELEMENTO EM Q11;
            IF ( REGISTRO DA TRANSMISSÃO DE NÍVEL
                        SUPERIOR DA IST ESTÁ HABILITADO )
            THEN REGISTRA EVENTO DE TRANSMISSÃO
                                        SUPERIOR COM SUCESSO;
         END;
      END;
   END;
END SUPDISPATCHER;
```

III.1.2.2 - OBSERVADOR LOCAL SUPERIOR - OLS

Este módulo tem a função de verificar o serviço da interface de nível superior das ISTs. Esta verificação deve ser realizada através da análise das sequências de UDs recebidas e geradas por estas interfaces , bem como pela análise do conteúdo das sondas correspondentes.

Ele é um módulo passivo, no sentido de que não retira nem

Ele é um módulo passivo, no sentido de que não retira nem insere UDs no teste, apenas analisa os que por ele circulam.

O OLS é específico para o algoritmo em teste e deve ser codificado com o auxílio do especialista neste algoritmo. Para a realização de testes manuais, a utilização do OLS (e dos demais observadores) é facultativa, entretanto é imperativo o seu uso para os testes automáticos.

Um "esqueleto" de implementação do OLS é apresentado abaixo.

```
PROCEDURE SUPLOCALOBSERVER;
/* Observador local superior */
BEGIN;
   WHILE ( EXISTE ELEMENTO EM Q12 )
   DO BEGIN;
      RETIRA ELEMENTO DE Q12;
      REALIZA PROCEDIMENTO DE OBSERVAÇÃO LOCAL SUPERIOR;
      /* MÁQUINA PREDICADO-TRANSIÇÃO */
      INSERE ELEMENTO EM Q10;
   END;
   WHILE ( EXISTE ELEMENTO EM Q11 )
   DO BEGIN;
      RETIRA ELEMENTO DE Q11;
      REALIZA PROCEDIMENTO DE OBSERVAÇÃO LOCAL SUPERIOR;
      /* MÁQUINA PREDICADO-TRANSIÇÃO */
      INSERE ELEMENTO EM Q1;
  END;
END SUPLOCALOBSERVER;
```

A estratégia de tornar todos as UDs do nível superior disponíveis para o OLS em filas concentradoras proporciona grande flexibilidade na sua implementação.

Através da seleção das UDs que são observadas e do uso de algoritmos convenientes podemos implementar diferentes configurações para o OLS :

. Ele pode se comportar como se houvesse um conjunto de OLS. Cada um deles específico para uma IST (que podem ser iguais ou não) e testando condições particulares a ela.

. Ele pode se comportar como um super-OLS, testando condições globais ao conjunto de interfaces superiores das ISTs, ao invés de condições particulares a cada uma delas.

. Uma combinação das configurações anteriores também pode ser implementada.

À deteção de falhas o OLS pode notificar o usuário através de registro de eventos, exteriorização de mensagens informativas no vídeo, emissão de sinais sonoros, etc.

III.1.2.3 - DESPACHANTE INFERIOR - DISPI

O DISPI é o análogo do DISPS para o nível inferior.

III.1.2.4 - NÍVEL INFERIOR - NI

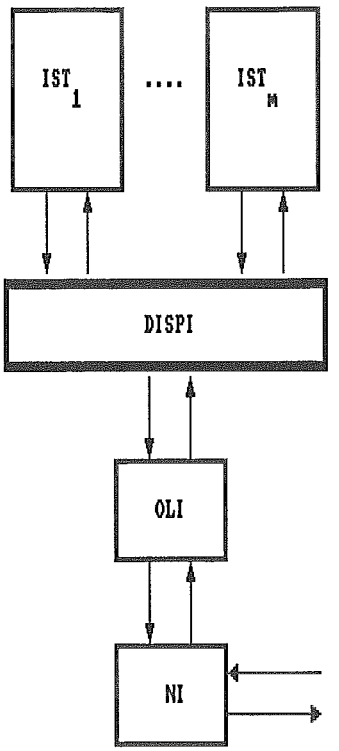
Este módulo provê o serviço do nível inferior do protocolo. Ele recebe as UDs geradas pelas ISTs através de seu nível inferior e gera as UDs correspondentes para as ISTs destino.

O NI é um bloco ativo, pois retira e insere UDs no fluxo de dados do GT. Ele é específico para o algoritmo em teste e deve ser codificado com o auxílio de seu especialista.

Existem duas opções de implementação do NI - figura III-4, fortemente relacionadas à implementação do OLI :

. O OLI está disposto entre o DISPI e o NI - figura III-4a e figura III-3. Neste caso o OLI é análogo ao OLS, suportando as mesmas opções deste. Esta é a opção preferencial, por ser mais abrangente, e será a adotada no

a)



b)

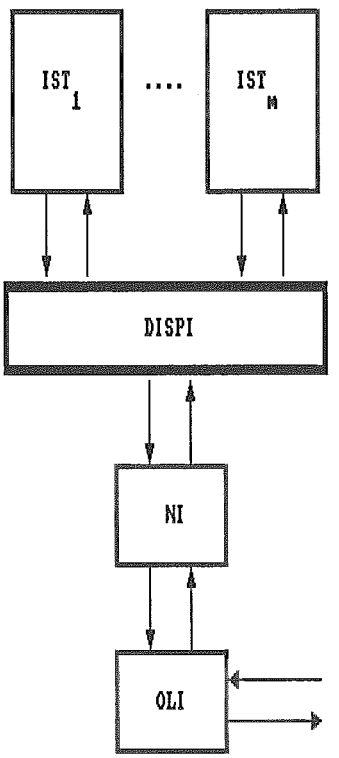


Figura III-4 - Opcoes de Implementacao do Conjunto NI/OLI

restante deste trabalho.

. O NI está entre o OLI e o DISPI - figura III-4b. Esta abordagem é mais limitada, posto que o NI introduz alguma incerteza para a deteção de erros por parte do OLI, de forma equivalente à arquitetura tradicional para teste remoto.

O NI - opção a - pode realizar o serviço do nível inferior à recepção das UDs das ISTs (sentido q6 -> q1), ou ao retorno destas (q5 -> q7) do restante do GT. Estas opções são indiferentes para o OLI mas têm reflexo em outros blocos do GT - CT e OG .

O NI é facultativo para testes manuais, mas seu uso facilita o trabalho do usuário, simplificando o conjunto de entradas que deve ser digitado. Para testes automáticos O NI é necessário.

Um "esqueleto" de implementação é apresentado abaixo.

```
PROCEDURE N1LEVEL;
/* Nivel (N-1) */
BEGIN;
   WHILE ( EXISTE ELEMENTO EM Q5 )
   DO BEGIN;
      RETIRA ELEMENTO DE Q5;
      REALIZA O SERVIÇO DO NÍVEL (N-1);
      /* MÁQUINA PREDICADO TRANSIÇÃO */
      INSERE O ELEMENTO EM Q7;
   END;
   WHILE ( EXISTE ELEMENTO EM Q6 )
   DO BEGIN;
      RETIRA ELEMENTO DE Q6;
      REALIZA O SERVIÇO DO NÍVEL (N-1);
      /* MÁQUNA PREDICADO TRANSIÇÃO */
      INSERE O ELEMENTO EM Q1;
   END;
END N1LEVEL;
```

III.1.2.5 - OBSERVADOR LOCAL INFERIOR - OLI

Se adotarmos a opção de implementação proposta no item anterior o OLI é o análogo do OLS para o nível inferior.

III.1.2.6 - OBSERVADOR GLOBAL - OG

A função do OG é verificar predicados globais ao conjunto de ISTs. As condições testadas pelo OG são de mais alto nível que as testadas pelos observadores locais, ou são condições que impliquem em analisar as UDs de ambos os níveis.

Uma vez que todas as UDs dos dois níveis estão concentradas em suas filas de entrada q1 e q3, o OG tem completo poder de detecção de erros (se todos os pontos de observação forem acessíveis), ele pode efetuar qualquer tipo de teste e inclusive englobar o trabalho dos outros observadores - OLI e OLS, muito embora por questões de modularidade e facilidade de programação isto não seja recomendável.

A atuação do OG no sentido q1 -> q2 corresponde à análise das UDs geradas pelas ISTs na varredura anterior; q3 -> q4 é a observação dos estímulos (UDs) inseridos pelo CT ou pelo usuário.

Como já foi discutido no item referente ao NI (item III.1.2.4) deve-se ainda atentar para a compatibilidade do OG com o conjunto NI/OLI.

Outro detalhe importante a ser previsto na construção do OG é o fato de que algumas UDs, notadamente as de nível inferior, são passadas adiante (q2 -> q3) pelo CT. O OG deve evitar observá-las duas vezes.

Um "esqueleto" de implementação é mostrado abaixo.

```
PROCEDURE GLOBALOBSERVER;
/* Observador global */
BEGIN;
   WHILE ( EXISTE ELEMENTO EM Q1 )
   DO BEGIN;
      RETIRA ELEMENTO DE Q1;
      REALIZA PROCEDIMENTO DE OBSERVAÇÃO GLOBAL;
      INSERE ELEMENTO EM Q2;
   END;
   WHILE ( EXISTE ELEMENTO EM Q3 )
   DO BEGIN;
      RETIRA ELEMENTO DE Q3;
      REALIZA PROCEDIMENTO DE OBSERVAÇÃO GLOBAL;
      INSERE ELEMENTO EM Q4;
   END;
END GLOBALOBSERVER;
```

III.1.2.7 - DESPACHANTE GLOBAL - DISPG

Este é simplesmente um elemento roteador. Sua função é distribuir as UDs de q4, onde estão concentradas UDs do nível superior e inferior, para as filas correspondentes ao seu nível. A cada chamada DESPG esvazia q4.

```
PROCEDURE GLOBALDISPTCHER;
/* Despachante global */
BEGIN;
   WHILE ( EXISTE ELEMENTO EM Q4 )
   DO BEGIN;
      IF ( ELEMENTO É DO NÍVEL SUPERIOR )
      THEN INSERE ELEMENTO EM Q12;
      ELSE INSERE ELEMENTO EM Q5;
   END;
END GLOBALDISPATCHER;
```

III.1.2.8 - CONDUTOR DE TESTE - CT

O CT é o responsável pela aplicação do teste. Ele é um módulo ativo, inserindo estímulos (UDs) para as ISTs através de q3 e recebendo as UDs geradas por elas em q2.

A IHM opera em paralelo com o CT e permite ao usuário atuar manualmente sobre as filas q2 e q3 para para inserir, retirar e modificar UDs para as ISTs.

O CT é específico para cada teste de determinado algoritmo e dependente do tipo de teste que se deseja realizar :

. TESTE MANUAL
O CT é facultativo para testes manuais. Entretanto é conveniente utilizar ao menos um CT elementar, que faça automaticamente o roteamento das UDs de nível inferior de q2 para q3 e descarte as UDs de nível superior (Uma vez na fila q2 estas UDs já foram registradas na lista de eventos, e analisadas pelos observadores, portanto podem ser descartadas), trabalho este que teria de ser realizado manualmente.

. TESTE AUTOMÁTICO ALEATÓRIO
O CT gera aleatoriamente um tráfego de UDs, inserindo-as em q3. Ele processa as UDs geradas pelas ISTs , descartando-as, se for o caso, ou enviando-as a seus destinos.

. TESTE AUTOMÁTICO POR SEQUENCIA DE TESTE
Existem duas opções :

a) O CT é adaptativo e gera "on-line" uma sequência de teste dependente da evolução das ISTs;

b) O CT é simplesmente o aplicador de uma sequência de teste prefixada, gerada "off-line" por algum processo (automático ou manual).

Um "esqueleto" de CT para o teste manual é mostrado abaixo.

```
PROCEDURE TESTCONDUCTOR;
/* Condutor de teste */
BEGIN;
   WHILE ( EXISTE ELEMENTO EM Q2 )
   DO BEGIN;
      RETIRA ELEMENTO DE Q2;
      IF ( ELEMENTO É DO NÍVEL SUPERIOR )
      THEN DEVOLVE ELEMENTO AO SACO;
      ELSE INSERE ELEMENTO EM Q3;
   END;
END TESTCONDUCTOR;
```

III.1.2.9 - FUNCIONAMENTO DO GT

Todos os blocos do GT, mostrados na figura III-3 e discutidos nos itens anteriores, são implementados como rotinas e utilizados pelo seu programa principal.

O programa principal do GT executa os demais blocos sequencialmente segundo o algoritmo abaixo :

```
PROCEDURE EXECUTETESTER;
BEGIN;
   TESTCONDUCTOR;     /* CT - Condutor de Teste */
   GLOBALOBSERVER;    /* OG - Observador Global */
   GLOBALDISPATCHER; /* DISPG - Despachante Global */
   N1LEVEL;           /* NI - Nivel (N-1) */
   INFLOCALOBSERVER; /* OLI - Observador Local Inferior */
   SUPLOCALOBSERVER; /* OLS - Observador Local Superior */
   INFDISPATCHER;     /* DISPI - Despachante Inferior */
   SUPDISPATCHER;     /* DISPS - Despachante Superior */
   /* Realização de uma varredura das ISTs */
   SUPDISPATCHER;     /* DISPS - Despachante Superior */
   INFDISPATCHER;     /* DISPI - Despachante Inferior */
   SUPLOCALOBSERVER; /* OLS - Observador Local Superior */
   INFLOCALOBSERVER; /* OLI - Observador Local Inferior */
   N1LEVEL;           /* NI - Nivel (N-1) */
   GLOBALOBSERVER;    /* OG - Observador Global */.
```

```
   /* Incremento do tempo virtual de simulação */
END EXECUTETESTER;
```

Uma chamada a EXECUTETESTER provoca um ciclo completo de fluxo de dados no interior do GT. No primeiro semiciclo os dados partem do CT e atingem as ISTs, no outro fazem o caminho inverso.

O primeiro semiciclo é definido em EXECUTETESTER pelo código desde a chamada a TESTCONDUCTOR até a primeira chamada a SUPDISPATCHER. Nele as UDs são inseridas em q3 pelo CT, onde se juntam às já inseridas pelo usuário através da IHM. Estas UDs são roteadas (DISPG , DISPS , DISPI, NI) e observadas (OG , OLI , OLS) pelos diversos blocos do testador, terminando nas filas de entrada das ISTs.

A seguir é realizada uma varredura das ISTs que consomem suas filas de entrada e eventualmente geram novas UDs em suas filas de saída.

Após a varredura inicia-se o segundo semiciclo, caracterizado pelo código desde a segunda chamada a SUPDISPATCHER até o fim de EXECUTETESTER. Nele as filas de saída das ISTs são esvaziadas e as UDs ali existentes circulam através dos blocos do testador sendo novamente roteadas e observadas. Estas UDs terminam em q2, onde ficam disponíveis para o CT e para a IHM. Após a ação destes, o ciclo recomeça.

Como em cada semiciclo os dados circulam em apenas um sentido de cada vez (CT -> ISTs ou ISTs -> CT), apenas as filas internas do CT correspondentes a este sentido são utilizadas. Os módulos (rotinas) do GT, chamadas uma vez a cada semiciclo, devem prever isto em seus algoritmos e examinar sempre as filas correspondentes aos dois sentidos, posto que a priori elas não sabem em que semiciclo o GT se encontra.

Esta é apenas uma opção de implementação, existem outras soluções possíveis : dividir os módulos do GT em duas rotinas, uma para cada sentido; passar parâmetros para as rotinas indicando o sentido de operação; uso de variáveis globais; etc.

Além disto todo o discurso acerca da implementação dos módulos do testador assume que eles esvaziam todas as suas filas de entrada a cada chamada. Isto garante fluxos unidirecionais de dados a cada semiciclo. Existe a possibilidade de que os módulos específicos do teste - NI, OLS, OLI, OG e CT - não atuem assim (mas é imperativo que eles evitem o crescimento indefinido destas filas). Isto causará fluxo bidirecional de dados e possívelmente introduzirá complicações adicionais nos algoritmos destes módulos, o que é, em princípio, desnecessário.

O dimensionamento da fatia-de-tempo atribuída pelo testador às ISTs pode distorcer o funcionamento do algoritmo em teste.

Tal distorção é devida à estratégia de execução do GT somente nos intervalos entre as varreduras. Isto faz com que a comunicação entre as ISTs (roteamento de UDs), bem como a aplicação das sequências de teste, sejam realizadas apenas nestes instantes discretos.

Se a fatia fôr superdimensionada então o testador pode introduzir errôneamente atrasos no funcionamento do algoritmo.

Um exemplo para ilustrar esta situação é apresentado abaixo.

. Supondo que o sistema-alvo do algoritmo em teste seja uma rede em que os tempos de transmissão e tratamento das mensagens são muito menores que a fatia-de-tempo utilizada.

. Supondo que a comunicação seja do tipo "stop-and-wait",

de forma que uma IST deve aguardar, após o envio de um quadro, a recepção de sua confirmação ("ack") para prosseguir.

. Se uma UD fôr transmitida por determinada IST no início da sua fatia então esta UD só atingirá a sua IST destino ao fim da varredura corrente, e só sera tratada por esta na próxima varredura.

. Uma vez que a IST destino trata o quadro e gera a confirmação instantâneamente, esta atingirá a IST origem da comunicação ao fim desta varredura.

. Assim a IST destino só receberá a resposta e poderá prosseguir a partir da segunda varredura a contar daquela em que enviou a mensagem.

. Portanto é necessário um mínimo de duas varreduras para concluir qualquer comunicação. Se a fatia está superdimensionada então tanto a IST fonte quanto a destino permanecem forçadamente ociosas em boa parte deste período.

. Tal situação não reflete o sistema-alvo, onde a mensagem seria, no intervalo correspondente a uma fatia, recebida pelo destinatário, tratada e respondida, habilitando a IST origem a prosseguir.

As distorções relacionadas ao dimensionamento da fatias tendem a desaparecer com a redução destas, cujo limite inferior é o tempo de execução de uma instrução do processador. Desta forma reduzimos a discretitude do tempo e nos aproximamos do caso contínuo.

O tempo T que o testador necessita para emular a execução de n ISTs durante um período t pode ser expresso pela seguinte fórmula :

$$T = n.t + (t/f).Tsim \qquad (1)$$

Onde :

. T - Tempo de execução do teste

. n - Número de ISTs

. t - Tempo de execução no sistema-alvo

. f - Fatia-de-tempo empregada. t/f expressa o número de varreduras necessárias para perfazer, no testador, o tempo t (para simplificar a análise assumimos que t é divisível por f).

. Tsim - Tempo médio gasto, por varredura, na execução do testador. Este compreende : o tempo de chaveamento das ISTs em uma varredura, que é fixo (em função de n); e o tempo gasto na execução do GT em uma varredura, que é variável e imprevisível (embora tenha uma parcela fixa, associada ao tratamento das filas internas do GT). Assim o tempo gasto pelo testador, por varredura, não é previsível, por isto adotamos um tempo médio (que é específico para cada teste).

A fórmula (1) supõe que as ISTs não realizam transferências espontâneas de processador (ver item III.2.2.1.4), o que pode redurir o tempo T. Portanto (1) pode ser tomada como limite superior.

Por (1) verificamos que à medida que a fatia é aumentada, tendendo para t, o tempo T, de teste, é reduzido, tendendo para n.t, que é o seu limite inferior.

Isto ocorre em função da redução do número de varreduras necessárias para perfazer o tempo t, o que reduz também o número de execuções do GT e o número de chaveamentos de ISTs. Portanto o "overhead" introduzido pelo testador diminui.

Por outro lado, à medida em que reduzimos a fatia, aumentamos o número de varreduras necessárias e por conseguinte o "overhead" do testador. Isto aumenta o tempo

T, gasto no teste.

Portanto concluímos que existe um compromisso entre a eficiência na realização dos testes (tempo gasto nos testes) e o grau de semelhança com a realidade obtido nestes.

Em princípio, o dimensionamento das fatias pode ser realizado empíricamente se dispusermos de dados relativos ao desempenho dos sistema-alvo (tempos de resposta, etc). Com isto podemos ajustar a fatia, na tentativa de aproximar destes valores os verificados no testador. A fatia empregada deve ser a maior que atenda a estas especificações, no intuito de otimizar o tempo gasto nos testes.

Os blocos do GT específicos para o protocolo em teste - OLS,OLI,NI,OG,CT - devem ser codificados com o auxílio do seu especialista. Estes blocos são rotinas cujas implementações apresentam as restrições já discutidas. Elas podem fazer uso de primitivas exportadas pelos demais módulos do testador.

Toda a "inteligência" do testador reside nestes módulos.
A decisão de separar a aplicação das sequências de teste (CT) da observação (OG, OLS, OLI) e do serviço do nível inferor (NI), representa uma abordagem estruturada para o teste. Todas estas tarefas poderiam ser realizadas pelo CT, mas isto não é recomendável pois este módulo seria complexo e seria necessário reescrevê-lo para cada teste.

Na abordagem proposta é necessário definir os observadores uma única vez, estes podem ser usados para diversos testes automáticos e/ou manuais. Os observadores não são específicos para cada teste e sim para o algoritmo em teste.

O mesmo vale com relação ao NI.
Já o CT pode ou não ser específico para cada teste. Se ele

fôr o aplicador de uma sequência de teste prefixada então ele pode ser uma ferramenta genérica, caso contrário (o CT gera entradas de forma aleatória ou adaptativa) será específico.

As propriedades do testador como gerador de um ambiente real de teste estão ligadas à codificação do modelo do nível inferior do algoritmo (NI) e do usuário do mesmo (CT), levadas em conta as restrições relacionadas à forma de exeução das ISTs e ao roteamento de UDs.

O teste simultâneo de várias ISTs, que não é usual, facilita a construção das sequências de teste, pois o conjunto de ISTs já apresenta um comportamento semelhante ao da operação real.

O testador é uma ferramenta genérica que permite o emprego de uma variedade de estratégias de teste. A figura III-3 mostra o testador completo. Como vimos ele deve ser configurado para cada teste, portanto, eventualmente alguns blocos não serão necessários e podem ser implementados por "dummies" (constituídos só pelos "esqueletos"), ou simplesmente eliminados (com os ajustes correspondentes na estrutura de filas e na rotina EXECUTETESTER).

III.1.3 - INTERFACE HOMEM MÁQUINA - IHM

As opções fornecidas pela IHM ao usuário são :

. Disparar e suspender a qualquer tempo a execução contínua de varreduras das ISTs;

. Comandar a execução de um número fixo de varreduras;

. Consultar/Alterar os seguintes atributos das ISTs :

a) Conjunto de endereços da interface superior ;

a) Conjunto de endereços da interface superior ;
b) Conjunto de endereços da interface inferior ;
c) Palavra de status, permitindo ativar ou desativar qualquer opção.

. Atuar sobre as UDs da fila q2 das seguintes formas :

a) Visualizar qualquer UD ;
b) Descartar qualquer UD ;
c) Despachar qualquer UD - Retirar a UD de q2 e inserir em q3;
d) Copiar qualquer UD para a área de edição.

. Editar UDs. Isto é feito em uma área de edição, correspondente a uma UD. São válidos os seguintes comandos de edição :

a) Visualizar o conteúdo corrente da área de edição;
b) Alterar o conteúdo da área de edição;
c) Despachar, inserindo uma cópia da área de edição em q3.

III.1.4 - REGISTRADOR DE EVENTOS - RE

O RE é simplesmente um gerador de "tracing" de recepção e transmissão de UDs. Ele não toma a iniciativa de registrar os eventos, mas apenas recebe solicitações de outros módulos, montando e registrando as cadeias de caracteres correspondentes às mensagens informativas de eventos.

III.2 - IMPLEMENTAÇÃO DO TESTADOR

Neste item primeiramente é feita uma descrição do "software" básico, e a seguir abordamos detalhes de implementação do testador.

III.2.1 - "SOFTWARE" BÁSICO

Esta descrição do "software" básico MODULA2 do CEPEL é bastante sucinta. Ela visa somente dar ao leitor o conhecimento necessário ao entendimento de detalhes da implementação do testador. Uma descrição detalhada foge ao escopo deste trabalho e pode ser encontrada na referência [20].

São descritas aqui apenas as principais primitivas, embora muitas outras sejam disponíveis.

III.2.1.1 - NÚCLEO MULTITAREFAS - NUCLEUS

O módulo NUCLEUS fornece o suporte para a programação concorrente no sistema de desenvolvimento MODULA2. Ele permite a criação de processos e provê mecanismos de sincronização e comunicação interprocessos.

Os processos são tarefas repetitivas e são implementados por rotinas executam infinitamente comandos de repetição. Eles podem ser criados a tempo de inicialização ou a tempo de execução, e, uma vez criados, não mais podem ser destruídos.

A comunicação interprocessos pode ser feita através de troca de mensagens por intermédio de caixas-postais ("mailboxes"). As caixas-postais são pontos de troca de mensagens que devem ser conhecidos tanto pelo processo emissor quanto pelo receptor. O mesmo vale com relação aos formatos das mensagens, que podem ser quaisquer.

São disponíveis primitivas bloqueadas e não bloqueadas para a troca de mensagens. Só é possível esperar em uma caixa-postal por vez, assim como só é possível enviar uma mensagem para uma caixa-postal por vez.

O NUCLEUS fornece ainda primitivas para o trabalho com

semáforos.

O tratamento de temporizações do NUCLEUS é feito por uma rotina instalada na interrupção de relógio do IBM-PC e tem uma taxa de 54msec. Sua estratégia de escalonamento ("scheduling") é baseada nas prioridades dos processos, que são estáticas e definidas por ocasião de suas criações.

As principais primitivas do NUCLEUS utilizadas no testador são as seguintes :

. CREATETASK (TASKCODE : PROC ; STACKSIZE , PRIORITY : CARDINAL);
Cria um processo, definido pela rotina TASKCODE, com área de pilha de STACKSIZE bytes e de prioridade PRIORITY. A área de pilha pode ter de 0 a 64Kbytes. A prioridade deve estar no intervalo [0..0FFFFH], sendo que 0 é a prioridade mais alta. CREATETASK deve ser chamada, a tempo de inicialização (antes da chamada a STARTTASKS) ou de execução , para a criação de todos os processos de aplicação. A criação de processos a tempo de inicialização habitualmente é feita no código de inicialização dos módulos a que eles pertencem.

.STARTTASKS;
Dispara o início do escalonamento. A sua chamada deve ser feita ao fim do procedimento de inicialização (ao fim do último código de inicialização a ser executado).

.CREATEMAILBOX (VAR MBX : MAILBOX);
Cria a caixa-postal MBX. Todas as caixas-postais devem ser criadas a priori de seu uso.

.SENDMSG (MBX : MAILBOX ; MSGADR : ADDRESS);
Envia a mensagem apontada por MSGADR para a caixa-postal MBX.

.WAITMSG(MBX:MAILBOX; DELAY:CARDINAL; VAR MSGADR:ADDRESS);

Espera no máximo um tempo DELAY, em unidades de 54msec (DELAY=0 => Espera bloqueada), pela chegada de uma mensagem na caixa-postal MBX. Se uma mensagem chegar no tempo estipulado, seu apontador é retornado em MSGADR, caso contrário é retornado o apontador vazio (NIL).

.ACCEPTMSG (MBX : MAILBOX ; VAR MSGADR : ADDRESS);
Busca uma mensagem na caixa-postal MBX. Se MBX possuir alguma mensagem pendente, esta é retirada e seu endereço é retornado em MSGADR, caso contrário é retornado o apontador vazio.

.TICK;
O "software" básico fornece uma facilidade de relógio interno para o registro do tempo virtual de simulação, em passos de 54msec . Este relógio é simplesmente um contador cujo incremento é feito através de chamadas a esta primitiva. São ainda disponíveis primitivas para a consulta a este relógio. (As primitivas relacionadas ao relógio de simulação não se encontram no NUCLEUS e sim no módulo TIMEDATE).

III.2.1.2 - PRIMITIVAS PARA CONSTRUÇÃO DE INTERFACES HOMEM/MÁQUINA - DIALOG

O DIALOG provê primitivas para exteriorização de dados em vídeo e para entrada de dados via teclado. Ele é voltado para a construção de diálogos para interfaces homem-máquina e faz uso de uma janela de vídeo dedicada.

Conceitualmente os diálogos são organizados em sequências, subdivididas em passos. Uma sequência de diálogo consiste na exteriorização de um conjunto de mensagens para o operador, e na leitura de um conjunto de campos digitados por este. Os campos podem ser de diversos tipos: numéricos, simbólicos, cadeias de caracteres, etc. A leitura de um campo constitui um passo de diálogo.
Uma sequência de diálogo é iniciada com a primitiva

"NEWDIALOG" e terminada com "DISPOSEDIALOG". Os passos para a leitura dos campos são chamadas a "APPENDXXX". O uso destas primitivas é esclarecido abaixo.

.NEWDIALOG;
Inicia uma nova sequência de diálogo.

.WTXT (TXT : ARRAY OF CHAR);
Exterioriza a cadeia TXT no vídeo;

.WLN;
Pula uma linha na janela de diálogo;

.APPENDDCA (LINF,LSUP : CARDINAL ; VAR RESULT : CARDINAL);
Inicia um passo de diálogo que consiste na leitura de um número natural no intervalo [LINF..LSUP], em representação decimal. O valor obtido é retornado em RESULT.

.APPENDHCA (LINF,LSUP : CARDINAL ; VAR RESULT : CARDINAL);
Semelhante ao anterior, mas o número é lido em representação hexadecimal.

.APPENDSTR (LEN : CARDINAL ; VAR RESULT : ARRAY OF CHAR);
Inicia um passo de diálogo para a leitura de uma cadeia de tamanho limitado por LEN, retornada em RESULT.

.APPENDSYM (VAR RANGE : ARRAY OF CHAR ; VAR RESULT : CARDINAL);
Inicia um passo para a leitura de um valor simbólico. Os símbolos válidos são definidos pela cadeia RANGE. Esta cadeia deve ser terminada por ".", com os símbolos separados por ",". O exemplo abaixo esclarece, definindo 3 símbolos :
RANGE : ARRAY [1..24] OF CHAR = "SYMBOL1,SYMBOL2,SYMBOL3.";
O resultado, retornado em RESULT, é o índice em RANGE do símbolo selecionado (o primeiro símbolo de RANGE tem índice 0). No exemplo acima se o usuário escolher "SYMBOL3" o valor retornado será 2.
.EXECUTE;

A chamada a EXECUTE realiza o atendimento ao teclado para a execução do diálogo de leitura dos campos relativos à sequência corrente de diálogo. Esta sequência é constituída pelos passos definidos desde o NEWDIALOG mais recente. Os diversos comandos de edição disponíveis tornam mais confortável o trabalho do usuário. EXECUTE pode ser chamada tantas vezes quantas for necessário realizar uma determinada sequência de diálogo.

.DISPOSEDIALOG;
Termina a sequência corrente de diálogo (que remonta ao último NEWDIALOG), descartando todos os seus passos e limpando a área correspondente da janela de diálogo. A sequência anterior, que remonta ao NEWDIALOG imediatamente anterior, torna-se novamente a sequência corrente.

III.2.1.3 - PRIMITIVAS PARA REGISTRO DE EVENTOS - LOGGER

O LOGGER permite a geração de históricos de eventos. Um evento consiste em uma cadeia de até 66 caracteres ASCII, rotulada com o relógio corrente e registrada no(s) dispositivo(s) selecionado(s) por ocasião de sua inserção.

Teclas dedicadas permitem mudar a qualquer instante o dispositivo corrente, direcionando o registro de eventos para a impressora, vídeo ou arquivos em disco.

Caso o dispositivo selecionado seja o vídeo os eventos são armazenados em um buffer circular interno, que contém os 25 mais recentes. Em uma janela de vídeo dedicada ao LOGGER são apresentados em tempo real os 10 eventos mais recentes. Comandos de teclado permitem percorrer o buffer circular interno do LOGGER e visualizar nesta janela o seu conteúdo.

Primitivas :

.LOG (FNT : LOGFNT ; VAR STR : ARRAY OF CHAR ; LEN : CARDINAL);

A cadeia STR, de tamanho LEN, é inserida como linha do histórico de eventos no estilo FNT. Para apresentação em vídeo, os estilos correspondem a cores (na impressora o tratamento pode ser feito de outra forma, mas isto não está implementado) Os estilos válidos estão no "set" LOGFNT e são :

a) NORMAL - Verde;
b) REVERSE - Amarelo;
c) BOLD - Vermelho;
d) ITALIC - Branco.

.UNQUEUE;
Os eventos são, após a inserção, armazenados em buffers intermediários internos ao LOGGER, sendo descarregados no dispositivo corrente somente à chamada de UNQUEUE.

III.2.1.4 - ROTINAS UTILITÁRIAS - UTILITY

O módulo UTILITY fornece um conjunto de rotinas utilitárias de propósito geral. Elas são relacionadas abaixo.

.CARDTODECSTR (CARD : CARDINAL ; BLANK : BOOLEAN ; VAR STR : ARRAY OF CHAR);
Converte um valor inteiro positivo - CARD - para uma cadeia de caracteres em representação decimal. BLANK indica se as posições não ocupadas à esquerda devem ser preenchidas com espaços (TRUE) ou zeros (FALSE). A cadeia gerada é retornada em STR.

.CARDTOHEXSTR (CARD : CARDINAL ; BLANK : BOOLEAN ; VAR STR : ARRAY OF CHAR);
Semelhante ao anterior, mas a cadeia é obtida em representação hexadecimal.

.INSERT (SUBSTR : ARRAY OF CHAR ; VAR STR : ARRAY OF CHAR ; VAR INX : CARDINAL);
É uma rotina auxiliar para manipulação de cadeias de caracteres. Insere a cadeia SUBSTR na cadeia STR, a partir

da posição INX. O valor de INX é retornado incrementado do tamanho de SUBSTR.

.RANDOM (N : CARDINAL) : CARDINAL;
Gera um número pseudo-aleatório no intervalo [0..N-1].

III.2.2 - TESTADOR

III.2.2.1 - NÚCLEO - NU

O número de alterações necessárias para executar implementações reais no testador é propositadamente pequeno. Para uma implementação ser utilizada como IST ela deve obedecer às seguintes restrições :

. Não deve realizar acessos a periféricos de E/S;
O acesso aos periféricos de E/S é privativo do testador. Os recursos disponíveis para as ISTs são : o processador e os serviços do testador.
As ISTs não podem realizar tratamento de interrupções (exceto interrupção de evento externo - ver módulo SERVIÇO) nem alterar o vetor de interrupções da UCP.
Considera-se, portanto, que os sistemas operacionais das ISTs são simples e modulares, basicamente núcleos multitarefas semelhantes ao NUCLEUS. O que é verdade para os nossos algoritmos reais (isto não é verdade somente para o nível aplicação, que corresponde aos usuários dos protocolos).
Evitamos o tratamento da convivência de diversos sistemas multitarefas simultâneamente exteriorizando em vídeo, acessando arquivos, etc.

. Não deve utilizar alocação dinâmica de memória;
A instalação de diversas ISTs simultâneamente residentes em memória é incompatível com os mecanismos de alocação dinâmica de muitos sistemas (p. ex. Sistema MODULA2), assim a área reservada para cada IST é estática e definida a tempo de instalação no testador.

. Deve realizar o procedimento de instalação no testador (NU);
Um dos serviços do testador é específico para a instalação das ISTs. O procedimento de inicialização das ISTs deve ainda incluir : a configuração dos endereços de suas interfaces; a configuração das rotinas de tratamento de relógio de seus núcleos multitarefas como iterrupções de eventos externos para o NU (ver módulo SERVIÇO).

. Deve ser dotada da interface adequada para o acesso aos serviços do testador.
Os procedimentos reais de transmissão e recepção de dados devem ser adaptados para o uso dos serviços do testador. Isto deve ser feito através de um módulo específico (INTERFACE).

As ISTs devem ser constituídas por módulos executáveis formato DOS - arquivos .EXE.

O NU não possui programa principal nem processos, ele é composto por 4 módulos, que exportam um conjunto de rotinas e estruturas de dados para o subsistema testador (além dos serviços para as ISTs) : FILAS, TAREFAS, SONDAS e SERVIÇO. Estes blocos serão detalhados a seguir.

III.2.2.1.1 - FILAS

Este módulo contem um conjunto de rotinas e estruturas de dados associadas à manipulação das filas internas do testador, cujos elementos são as UDs.

Todas as filas do sistema apresentam estrutura idêntica, do tipo FIFO e simplesmente encadeadas. O formato da UD é apresentado abaixo.

```
ELEMPTR : POINTER TO ELEM;
```

```
ELEM : RECORD
          NEXT : ELEMPTR;
          LEVEL : LEVELSET;
          DST : CARDINAL;
          SUBNET : CARDINAL;
          LEN : CARDINAL;
          DATA : ARRAY OF CHAR;
       END;
```

Onde os campos tem o seguinte significado :

. NEXT - Apontador para a próxima UD da fila.

. LEVEL - Indica o nível da UD. Os níveis válidos estão no "set" LEVELSET e são os seguintes :

a) SUPLEVEL - Nível superior;
b) INFLEVEL - Nível inferior.

Uma UD não pode pertencer simultâneamente aos dois níveis.

. DST - Endereço destino da UD. Os endereços válidos estão no intervalo [0..15]. Uma UD é recebida (se for o caso) por todas as ISTs em que estiver "setado" (em "um") o bit correspondente ao seu endereço nas palavras de endereçamento (das ISTs) correspondentes ao seu nível. Ver a discussão do endereçamento das interfaces das ISTs no próximo item - III.2.2.1.2.

. SUBNET - Campo auxiliar de endereçamento de UDs. Sua utilização é análoga ao campo DST, e seu objetivo é fornecer facilidades adicionais como : simular duas redes que se comunicam através de uma ponte ou "gateway"; simular uma falha que cause ilhamento de parte de uma rede, etc.

. LEN - Tamanho da área de dados.

. DATA - Área de dados.

A estrutura da fila é a seguinte :

```
FILA = RECORD;
          HEAD : ELEMPTR;
          TAIL : ELEMPTR;
       END;
```

Os elementos de fila são disponíveis, em número limitado, em um saco ("pool" de áreas livres). A sua utilização requer a obtenção, a priori, de um elemento vago, e sua posterior devolução ao saco.

Para a realizar uma transmissão, a IST deve obter uma UD vaga, preenchê-la e inserir a mesma na fila de saída correspondente. Esta UD será então roteada para o(s) seu(s) destino(s) que deverão retirá-la de suas filas de entrada, utilizá-las e a seguir devolvê-las ao saco. A obtenção de UDs vagas, seu preenchimento, recepção, envio e retorno ao saco são as funções a serem realizadas pelas interfaces específicas para teste, da qual devem ser dotadas todas as ISTs.

As primitivas exportadas por este módulo são as seguintes :

. GETELEM (VAR FL : FILA ; VAR ELPTR : ELEMPTR);
Retira a primeira UD de FL e retorna seu apontador em ELPTR.

. PUTELEM (VAR FL : FILA ; VAR ELPTR : ELEMPTR);
Insere a UD apontada por ELPTR no final de FL.

. EXTRACTELEM (VAR FL : FILA ; VAR ELPTR : ELEMPTR);
Retira de FL a UD apontada por ELPTR.

. RESERVELEM (VAR ELPTR : ELEMPTR);
Obtem do saco uma UD vaga e retorna o seu apontador em ELPTR.
. RELEASELEM (VAR ELPTR : ELEMPTR);
Devolve a UD apontada por ELPTR ao saco.

. COPYELEM (VAR SRCPTR , DSTPTR : ELEMPTR);
Copia o conteúdo da UD apontada por SRCPTR para a UD apontada por DSTPTR.
Os algoritmos destas rotinas são elementares. Algumas delas são disponíveis somente para o testador, enquanto as demais o são também para as ISTs (através do módulo SERVIÇO).

III.2.2.1.2 - TAREFAS

O NU armazena e manipula as informações de controle relevantes acêrca das ISTs através de uma tabela contendo os seus descritores. O módulo TAREFAS contém as declarações das estruturas de dados e rotinas associadas a esta tabela.

A tabela de descritores é implementada como uma fila simplesmente encadeada, sendo que cada entrada tem o seguinte formato :

```
JOBDSCPTR : POINTER TO JOBDSC;

JOBDSC = RECORD;
            NEXT : JOBDSCPTR;
            IDENT: CARDINAL;
            PORTI , PORTS : BITSET;
            STATUS : JOBSTATUSET;
            QSUPIN , QSUPOUT , QINFIN , QINFOUT : FILA;
            EXTEVEHDLADD : ADDRESS;
            INTMASK : BITSET ;
            STACKSEG , STACKOFS : WORD;
            RXINFERCNT , TXINFERCNT : CARDINAL;
            RXSUPERCNT , TXSUPERCNT : CARDINAL;
         END;
```

Onde :

. NEXT - Apontador para o próximo descritor;

. IDENT - Número que identifica unívocamente uma IST;

. PORTI - Conjunto de endereços da interface inferior da IST. Cada bit deste conjunto ("BITSET" - conjunto de bits) corresponde a um endereço, assim são possíveis 16 endereços (de 0 a 15). As ISTs podem possuir qualquer subconjunto destes endereços.

. PORTS - Conjunto de endereços da interface superior da IST. O endereçamento desta interface é idêntico ao do nível inferior.

. STATUS - Palavra de status da IST. Esta palavra controla a habilitação da transmissão e recepção das interfaces das ISTs, bem como a habilitação dos registros automáticos dos eventos de transmissão e recepção através destas interfaces. Os status válidos estão no "set" JOBSTATUSET e são os seguintes :

a) RXINFDOWN - Desabilita a recepção através do nível inferior.
b) TXINFDOWN - Desabilita a transmissão através do nível inferior.
c) RXSUPDOWN - Desabilita a recepção através do nível superior.
d) TXSUPDOWN - Desabilita a transmissão através do nível superior.
e) RXINFLOG - Habilita o registro automático de eventos de recepção através do nível inferior.
f) TXINFLOG - Habilita o registro automático de eventos de transmissão através do nível inferior.
g) RXSUPLOG - Habilita o registro automático de eventos de recepção através do nível superior.
f) TXSUPLOG - Habilita o registro automático de eventos de transmissão através do nível superior.

Qualquer destas opções pode ser selecionada isoladamente.
. QSUPIN , QSUPOUT , QINFIN , QINFOUT - Filas de entrada (IN) e saída (OUT) das interfaces de nível superior

(SUP) e inferior (INF) das ISTs.

. EXTEVEHDLADD , INTMASK , STACKSEG , STACKOFS - Contexo (de baixo nível) das ISTs para uso do NU (módulo SERVIÇO).

. RXINFERCNT , TXINFERCNT, RXSUPERCNT, TXSUPERCNT - Contadores de erros de transmissão (TX) e recepção (RX) das interfaces de nível superior (SUP) e inferior (INF) das ISTs. Estes contadores são incrementados quando a operação correspondente é solicitada e está desabilitada pela palavra de status. Neste caso a UD correspondente é descartada. Estes campos são apenas campos informativos para o usuário e podem ser consultados através da IHM.

O formato da tabela de descritores é :

```
JOBQU = RECORD
            HEAD : JOBDSCPTR;
            TAIL : JOBDSCPTR;
        END;
```

O módulo TAREFAS exporta ainda duas rotinas :

. CREATEJOB (SS , SP : WORD);
Insere um novo descritor de IST na tabela do testador. As ISTs e o testador têm acesso a esta rotina através do módulo SERVIÇO (serviço de instalação de ISTs). Os parâmetros SS s SP estão relacionados ao contexto de baixo nível da IST.

. SORTJOBS (VAR JOBQUEUE : JOBQU);
Rearruma aleatóriamente a tabela JOBQUEUE, de descritores de ISTs. É utilizada pelo módulo SERVIÇO na realização das varreduras das ISTs.

III.2.2.1.3 - SONDAS

O mecanismo de sondas provê o acesso do testador às áreas de dados das ISTs.

O módulo SONDAS é um divulgador de endereços de sondas. Sua filosofia de utilização é a seguinte :

. As ISTs devem publicar, em princípio a tempo de inicialização, os endereços das sondas que lhe pertencem. As sondas são estáticas, isto significa que uma vez inicializadas não mais podem ser retiradas ou ter seus endereços modificados.

. Os módulos do testador obtêm estes endereços, a princípio também a tempo de inicialização, e passam a utilizá-los no decorrer de todo o teste, uma vez que são estáticos. O uso das sondas é livre, pode-se testar o seu conteúdo, exteriorizá-lo em vídeo, alterá-lo (perigoso !), etc.

As sondas são gerenciadas internamente através de uma tabela, cuja entrada tem o seguinte formato :

```
PROBEDSC = RECORD
              NAME : PROBENAMETYPE;
              ENDER : ADDRESS;
              LEN : CARDINAL;
           END;
```

Onde :

. NAME - As sondas são identificadas através de um nome lógico constituído por uma cadeia de caracteres do tipo PROBNAMETYPE : ARRAY [1..20]OF CHAR. A tabela de sondas é uma lista sequencial ordenada a partir deste campo.

. ENDER - Endereço da sonda.

. LEN - Tamanho, é o número de posições de memória ocupadas pela sonda.

As primitivas disponíveis são :

. CREATEPROBE (NAM : PROBNAMETYPE ; END : ADDRESS ; TAM : CARDINAL) : BOOLEAN;
Insere na tabela a sonda de nome NAM, endereço END e tamanho TAM. A rotina retorna FALSE caso a sonda já exista e neste caso ela não é inserida, caso contrário a inserção é realizada e CREATEPROBE retorna TRUE.

. GETPROBE (NAM : PROBNAMETYPE ; ENDPTR : POINTER TO ADDRESS ; TAMPTR : POINTER TO CARDINAL) : BOOLEAN;
Se a sonda de nome NAM pertencer à tabela então seu endereço é retornado na variável apontada por ENDPTR e seu tamanho na variável apontada por TAMPTR. Neste caso GETPROBE retorna TRUE, caso contrário FALSE.

CREATEPROBE é disponível para as ISTs através do módulo SERVIÇO. GETPROBE é exportada para o testador.

Para o uso deste mecanismo o nome lógico deve ser conhecido tanto pelo proprietário da sonda quanto por seus usuários. O mesmo vale com relação à interpretação de seu conteúdo.

III.2.2.1.4 - SERVIÇO

Este módulo provê o conjunto de primitivas acessíveis às ISTs e fornece ao testador a primitiva de execução de varreduras. O SERVIÇO realiza todo o processamento de baixo nível do NU.

O acesso das ISTs aos serviços do NU é feito através de uma interface de baixo nível - vetor de interrupções, registradores da UCP, pilha, etc. As interfaces das ISTs para o teste devem seguir este padrão. O SERVIÇO faz uso das primitivas dos demais módulos do NU para o tratamento destas solicitações.

São 13 os serviços disponíveis :

. GOCMD;
Comanda o início da simulação, ou seja, dispara o início da execução do subsistema testador. Este deve ser utilizado somente uma vez, a tempo de inicialização, após a instalação do testador e de todas as ISTs (ver próximo comando).

. INSCMD (TAM_MEMÓRIA : CARDINAL);
É necessário que todas as ISTs (e o subsistema testador) solicitem, a tempo de inicialização, a sua instalação no NU, o que é feito através deste comando. Todos os subsistemas devem ser instalados neste tempo, pois não é admitida a sua criação dinâmica. Na instalação de um subsistema o SERVIÇO suspende a sua execução, salva o seu contexto para posterior retomada de processamento e insere o descritor correspondente na tabela de descritores do NU. A instalação deve especificar TAM_MEMÓRIA, que é a área de memória ocupada pela IST em parágrafos (unidades de 16 bytes). O NU assume que a primeira instalação é sempre do subsistema testador. As instalações subsequentes são das ISTs.

. INCPRTCMD (ADD : CARDINAL ; LVL : LEVELSET);
Inclui ADD no conjunto de endereços do nível LVL da IST. Este comando (e o próximo) permitem às ISTs configurar dinâmicamente os endereços de suas interfaces.

. EXCPRTCMD (ADD : CARDINAL ; LVL : LEVELSET);
Exclui ADD do conjunto de endereços do nível LVL da IST.

. SETEXTEVEHDLCMD (ENDER : ADDRESS);
Permite às ISTs fornecer ao SERVIÇO o endereço ENDER de uma rotina de interrupção que será ativada pelo testador segundo períodos preestabelecidos. Esta facilidade fornece às ISTs um mecanismo para a instalação das rotinas de tratamento de relógio dos seus núcleos mutitarefas.
É necessário que os períodos de ativação sejam múltiplos da fatia-de-tempo. Eles devem ser especificados em número de

fatias (de quantas em quantas fatias o testador deve gerar esta interrupção), e devem ser dimensionados de forma a ficar o mais próximo possível dos intervalos utilizados pelos núcleos das máquinas do sistema-alvo.

. RSVCMD (ELPTR : ELEMPTR);
Obtém uma UD vaga, cujo endereço é retornado em ELEMPTR.

. RLSCMD (ELPTR : ELEMPTR);
Devolve ao saco uma UD, apontada por ELEMPTR, não mais necessária.

. SNDCMD (ELPTR : ELEMPTR ; LVL : LEVELSET);
Insere a UD apontada por ELEMPTR no final da fila de saída de nível LVL da IST. A UD deve estar devidamente preenchida.

. RCVCMD (ELPTR : ELEMPTR ; LVL : LEVELSET);
Retira a UD do início da fila de entrada de nível LVL da IST e retorna seu endereço em ELEMPTR.

. LOGCMD (CADEIAPTR : ADDRESS ; NBYTES : CARDINAL ; DYE : LOGFNT);
A cadeia de caracteres apontada por CADEIAPTR, de tamanho NBYTES, é inserida na lista de eventos, na cor DYE. Esta facilidade permite às ISTs exteriorizar mensagens informativas na lista de eventos. Este comando atenua a restrição de E/S das ISTs, pois permite a elas se comunicarem com o usuário.

. GETIDENTCMD (IDENTPTR : POINTER TO CARDINAL);
Retorna, na variável apontada por IDENTPTR, a identificação da IST (campo IDENT de seu descritor).

. CRTPRBCMD (NOMPTR , END , TAM);
Insere na tabela de sondas a sonda cujo nome é a cadeia apontada por NOMPTR, cujo endereço é ENDER e que ocupa TAM posições de memória.

. TRANSFERCMD;
Este serviço fornece às ISTs uma forma de acelerar o teste. Sua chamada encerra a fatia-de-tempo da IST corrente (sem completá-la) e passa o processador para a próxima IST, atribuindo-lhe uma nova fatia-de tempo. Quando uma IST conclui que não tem mais processamento a realizar, ela deve fazer esta transferência, pois somente na próxima fatia-de-tempo pode ocorrer a chegada de novas UDs e uma nova interrupção de evento externo.

A rotina usada pelo subsistema testador (GT) para comandar as varreduras das ISTs é exportada pelo SERVIÇO :

. DOAPLJOBSWAP;
Realiza uma varredura das ISTs, atribuindo fatias de tempo a cada uma, em ordem aleatória. DOAPLJOBSWAP ativa a rotina de tratamento de interrupção de relógio do SERVIÇO, instalada na interrupção de relógio do IBM-PC, que realiza o escalonamento das ISTs.

Os formatos das chamadas aos serviços fornecidos para as ISTs são aqui apresentados como rotinas para evitar os detalhes de baixo nível da sua implementação.

Os serviços exportados pelo NU para o subsistema testador são simplesmente rotinas públicas.

III.2.2.2 - GERENCIADOR DE TESTE - GT

O programa principal do GT é o processo PROCTESTER, que é o responsável pela recepção e tratamento dos comandos do usuário, recebidos da IHM.

Os comandos do GT são :

. Disparo de teste
À recepção deste comando o GT passa a realizar varreduras indefinidamente (O GT repete indefinidamente a execução da rotina EXECUTETESTER). Ao mesmo tempo ele aguarda novos comandos da IHM.

. Suspensão de teste
O GT termina a varredura corrente, suspende a execução de varreduras e passa a aguardar novos comandos da IHM.

. Disparo de um número fixo de varreduras
O GT dispara a realização de um número fixo de varreduras, definido pelo usuário, ao mesmo tempo em que aguarda novos comandos da IHM.

. Consulta/alteração
Uma parte da IHM, a consulta/alteração, está implementada no próprio PROCTESTER (isto será discutido no próximo item -III.2.2.3). À recepção deste comando o GT permite a consulta/alteração dos atributos das ISTs e das UDs de q2 e q3.

O PROCTESTER interage com a IHM através de troca de mensagens. Sua caixa-postal de entrada chama-se TESTERMBX .

O formato das mensagens para o GT é o seguinte :

```
TESTERMSGTYPE = RECORD
                    xxx : ADDRESS;
                    ACTION : TESTERACTIONSET;
                    PARAM : CARDINAL;
                    RSPMBX : MAILBOX;
                    STATUS : TESTERSTATUSET;
                END;
```

onde :

. XXX - Campo para uso do "software" básico.

. ACTION - Ação solicitada. As ações válidas, correspondentes aos comandos aceitos pelo GT, estão no "set" TESTERACTIONSET e são as seguintes :

a) TRIGGERCMD - Disparo de teste.
b) SUSPENDCMD - Suspensão de teste.
c) INTERACTCMD - Disparo de n varreduras. O parâmetro n é contido no campo PARAM.
d) CONSULTCMD - Consulta/alteração. Este comando só é aceito se o teste estiver suspenso.

. PARAM - Parâmetro para o comando INTERACTCMD;

. RSPMBX - Caixa-postal para onde deve ser enviada a resposta ao comando. A resposta é a própria mensagem recebida, com o campo status preenchido convenientemente.

. STATUS - Resultado do comando. Os status válidos estão no "set" TESTERSTATUSET e são os seguintes :

a) SUCCESS - Comando aceito e executado;
b) FAIL - Comando Inválido.

O PROCTESTER é uma máquina de estados em que as transições são disparadas pelas mensagens da IHM, de acordo com o diagrama de estados da figura III-5. O algoritmo deste processo é apresentado abaixo.

```
PROCEDURE PROCTESTER;
VAR ESTADO : ( SUSPENDED , TRIGGERED , INTERACT );
BEGIN;
   ESTADO = SUSPENDED;
   WHILE TRUE DO BEGIN; /* Forever */
      IF ( ESTADO <> SUSPENDED )
      THEN BEGIN;
         ACCEPT ( COMANDO ) ;
         IF ( RECEBEU UM COMANDO )
```

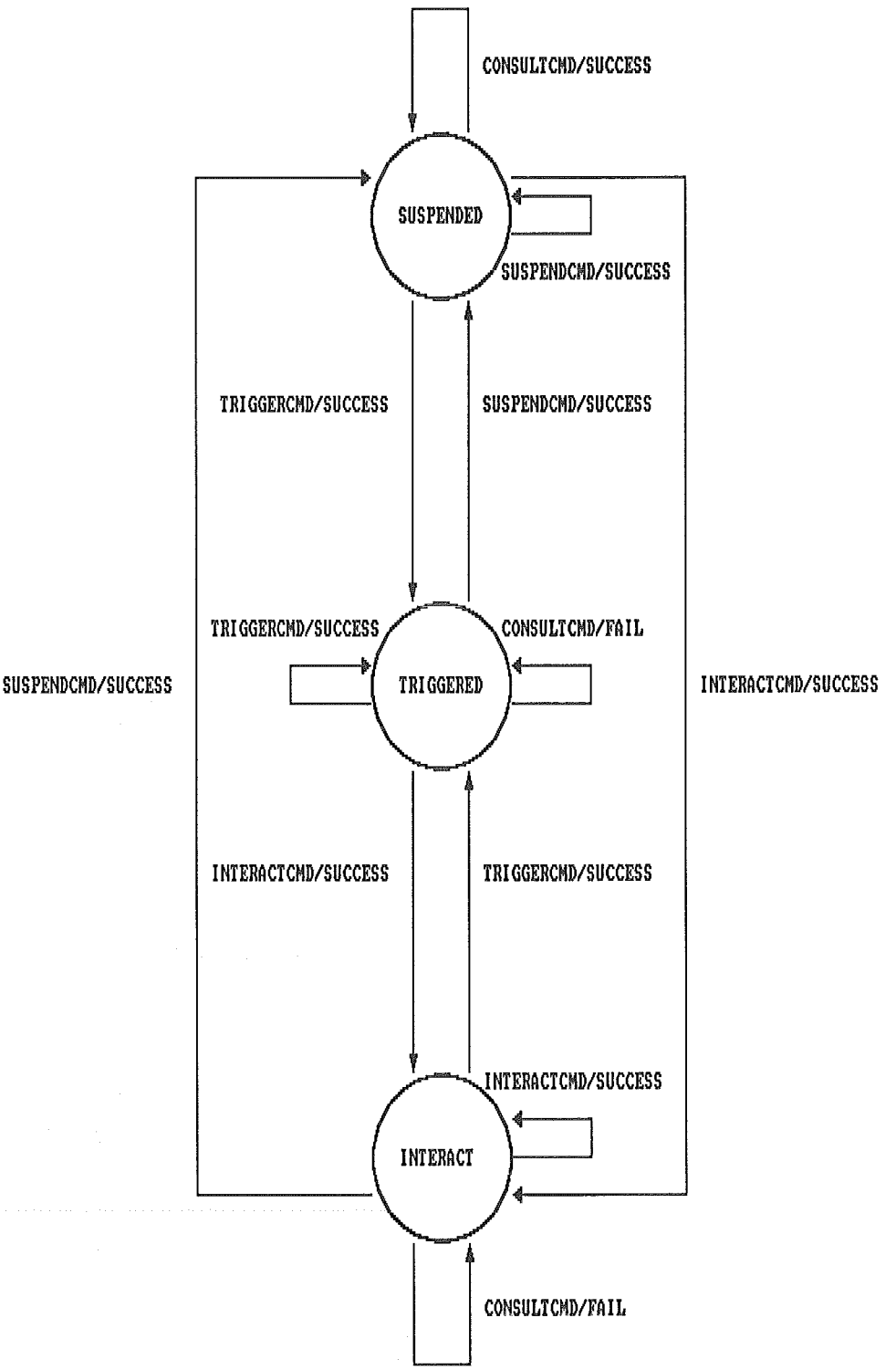


Figura III-5 - Diagrama de Estados do PROCTESTER

```
   THEN BEGIN;
      IF ( COMANDO = CONSULTACMD )
      THEN ENVIA ( FAIL );

      ELSE BEGIN;
         ESTADO = COMANDO;
         IF ( COMANDO = INTERACTCMD )
         THEN INICIALIZA CONTADOR DE INTERAÇÕES;
         ENVIA ( SUCCESS );
      END;
   END;
   ELSE BEGIN;
      EXECUTETESTER;
      IF ( ESTADO = INTERACT )
         AND
         ( ATINGIU O NUMERO DE INTERAÇÕES )
      THEN ESTADO = SUSPENDED;
   END;
END;
ELSE BEGIN;
   WAIT COMANDO;
   IF ( COMANDO = CONSULTCMD )
   THEN BEGIN;
      FAZ CONSULTA/ALTERAÇÃO;
      ENVIA ( SUCCESS );
   END;
   ELSE BEGIN;
      IF ( COMANDO = TRIGGERCMD )
         OR
         ( COMANDO = SUSPENDCMD )
         OR
         ( COMANDO = INTERACTCMD )
      THEN BEGIN;
         ESTADO = COMANDO;
         IF ( COMANDO = INTERACTCMD )
         THEN INICIALIZA CONTADOR DE INTERAÇÕES;
         ENVIA ( SUCCESS );
      END;
      ELSE ENVIA ( FAIL );
```

```
        END;
      END;
   END; /* Forever */
END PROCTESTER;
```

III.2.2.3 - INTERFACE HOMEM/MÁQUINA - IHM

Na figura III-1 a IHM foi apresentada como um bloco independente. Esta é apenas uma representação funcional. A implementação não é realizada desta forma, pois a parte referente à consulta/alteração é realizada dentro do PROCTESTER.

A decisão de partir a interface homem/máquina desta forma deve-se ao fato de que a consulta/alteração é uma parte muito específica do GT , pois é intimamente ligada às suas estruturas de dados. As outras opções de implementação implicam ou em uma taxa muito grande de troca de informações entre o GT (PROCTESTER) e a IHM, ou em possibilitar o acesso da IHM às estruturas internas do GT. Estas duas soluções não são satisfatórias, a primeira por requerer um algoritmo muito complexo e a segunda por ferir a modularidade do testador.

Atualmente o único bloco a receber comandos da IHM é o GT, entretanto futuras extensões do testador certamente irão introduzir novos blocos, modificar os já existentes e gerar a necessidade de incrementar a interface homem-máquina. A arquitetura proposta para a IHM é modular e suporta estas extensões.

Por isso é conveniente a prática de colocar apenas a parte de mais alto nível no PROCIHM (programa principal da IHM) e a parte mais específica no interior dos blocos associados.
O PROCIHM deverá fazer a parte preliminar do diálogo, distribuir mensagens contendo os comandos digitados pelo usuário para as caixas-postais de entrada dos demais blocos

e aguardar as respostas na(s) caixa(s)-postal(is) de resposta convenientes - figura III-6. Esta arquitetura permite alto grau de independência entre os módulos do testador.

O algoritmo do PROCIHM é apresentado abaixo :

```
PROCEDURE PROCIHM;
BEGIN;
   WHILE TRUE DO BEGIN; /* Forever */
      LE COMANDO DO TECLADO;
      IF ( COMANDO = TRIGGERCMD )
         OR
         ( COMANDO = SUSPENDCMD )
         OR
         ( COMANDO = CONSULTACMD )
      THEN BEGIN;
         MONTA MENSAGEM;
         ENVIA ( COMANDO ) ; /* Envia para PROC_TESTER */
         WAIT ( RESPOSTA ) ; /* Espera a resposta */
         IF ( STATUS DA RESPOSTA <> SUCCESS )
         THEN EXTERIORIZA MENSAGEM INFORMATIVA DE FALHA;
      END;
      ELSE BEGIN;
         IF ( COMANDO = INTERACT )
         THEN BEGIN;
            LE O NUMERO DE INTERAÇÕES DO TECLADO;
            MONTA MENSAGEM;
            ENVIA (COMANDO) ; /* Envia para PROC_TESTER */
            WAIT ( RESPOSTA ) ; /* Espera a resposta */
            IF ( STATUS DA RESPOSTA <> SUCCESS )
            THEN EXTERIORIZA MENSAGEM INFORMATIVA DE FALHA;
         END;
         ELSE EXERIORIZA MENSAGEM INFORMATIVA DE FALHA;
      END;
   END; /* Forever */
END PROCIHM;
```

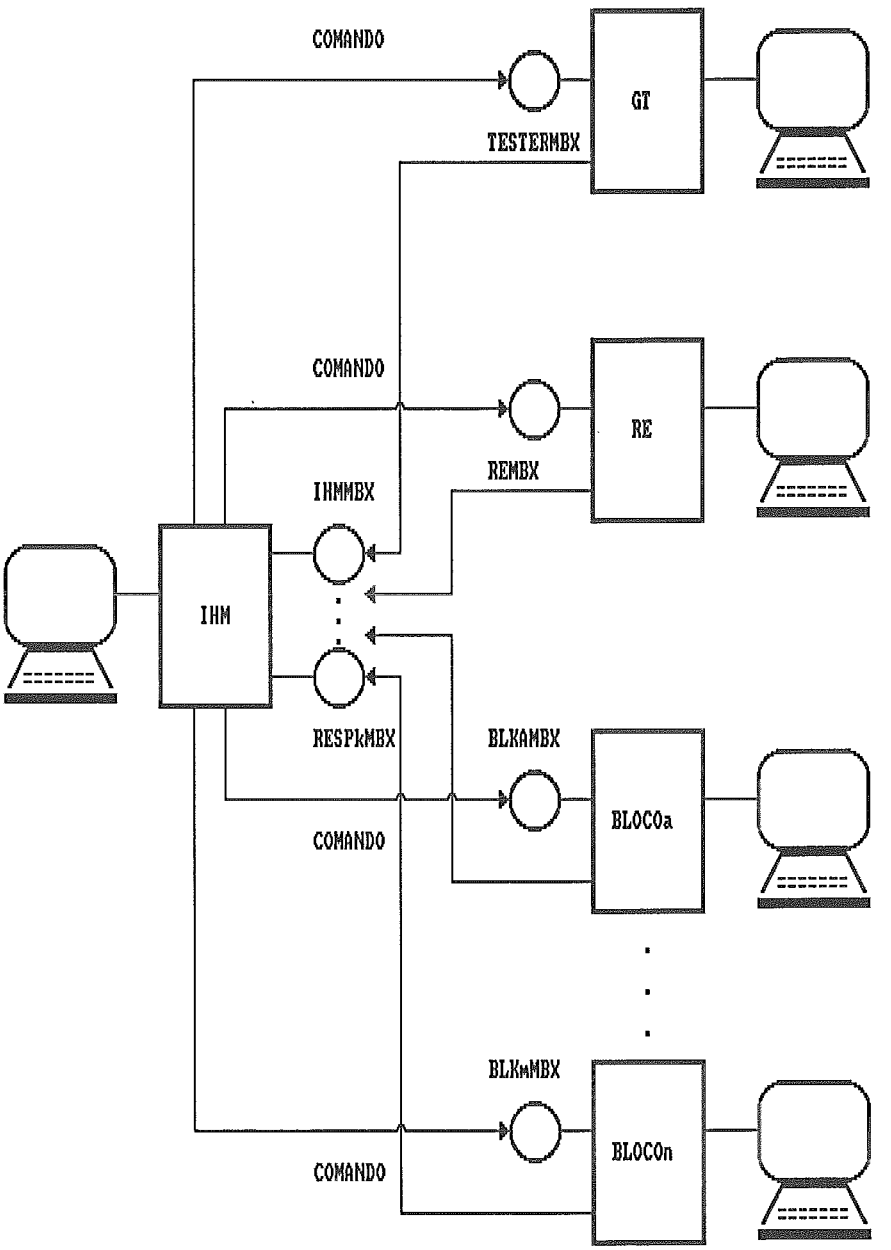


Figura III-6 - Arquitetura da IHM

III.2.2.4 - REGISTRADOR DE EVENTOS - RE

Atualmente o RE suporta apenas opções para registrar transmissões, recepções e perdas de UDs por parte das ISTs. Ele é utilizado somente pelos módulos NI e DISPS do GT. O RE é simplesmente um usuário do LOGGER do "software" básico.

O RE exporta uma única rotina :

. LOGMSG (VAR ELPTR : ELEMPTR ; IDENT : CARDINAL ; TIPO : TYPLOGSET);

Cujos parâmetros têm o seguinte significado :

. ELPTR - Apontador para a UD cujo evento deve ser registrado;

. IDENT - Identificação da IST relacionada ao evento;

. TIPO - Tipo de evento. Os tipos válidos estão no "set" TYPLOGSET e são os seguintes :

a) TXSUC - Transmissão com sucesso;
b) TXFAIL - Falha na transmissão;
c) RXSUC - Recepção com sucesso;
d) RXFAIL - Falha na recepção;

O formato das mensagens de evento é mostrado na figura III-7. Os eventos TXSUC são registrados na côr amarela, RXSUC em verde , TXFAIL e RXFAIL em vermelho.

```
hh:mm:ss.xxx III TTTT L Dst=>DD  Sub=>SS   Len=>LL   Cont=>BB ..... BB
                                                      01 ..... LENGHT
```

hh:mm:ss.xxx - relogio inserido pelo LOGGER

III (3 caracteres) - identificacao da IST

TTTT (4 caracteres) - tipo de evento: TXOF - transmissao com sucesso
TXFL - falha na transmissao
RXOF - recepcao com sucesso
RXPL - falha na recepcao

L (1 caracter) - nivel: I - inferior
S - superior

DD (2 caracteres): endereco destino da PDU

SS (2 caracteres): endereco da subrede

LL (2 caracteres): tamanho da mensagem

BBs => bytes da mensagem

Figura III-7 - Formato das Mensagens de Eventos

CAPÍTULO IV

REALIZAÇÃO DE TESTES

CAPÍTULO IV

Este capítulo relata experiências de utilização do testador como ferramenta de auxílio ao desenvolvimento de algoritmos distribuídos.

Tais experiências compreenderam testes de diversos algoritmos, com dois objetivos : a depuração do testador e a avaliação de seu desempenho como ferramenta.

A bateria de testes incluiu :

. Um algoritmo elementar, no qual foi realizado teste manual;

. Um protocolo de comunicação ponto-a-ponto, proposto por TANENBAUM [21] (cap4; sec4.2; pag 158), no qual foi realizado teste manual e automático;

. Um algoritmo distribuído relacionado à topologia de redes, proposto por SEGALL [22] (pp 28,29), no qual foi realizado teste manual e automático.

Tais testes serão detalhados nos próximos itens.

As ISTs que implementam os algoritmos usados nos testes foram desenvolvidas no sistema de desenvolvimento MODULA2 e utilizam o "software" básico CEPEL-MODULA2.

IV.1 - PRIMEIROS TESTES - DEPURAÇÃO

Os primeiros testes visavam basicamente a depuração do testador. Para isto era necessário exercitar :

. Os serviços por ele exportados;

. As varreduras de ISTs;

. A geração de interrupções de eventos externos;

. Os recursos da IHM.

A implementação destes itens (exceto a IHM) é realizada no módulo SERVIÇO do NÚCLEO do testador. O SERVIÇO faz uso intensivo de recursos de baixo nível, dependentes do hardware e do sistema MODULA2, como : "inline code"; pilha e registradores da UCP; portas de E/S; acesso a microperiféricos; etc. Por isso a sua depuração foi bastante penosa e demorada.

A IST que foi utilizada nesta etapa é implementada por um programa que não faz uso do "software" básico CEPEL-MODULA2. Ele necessita somente do sistema em tempo real MODULA2 (RTS -"Run Time System"). Seu algoritmo é apresentado a seguir :

```
PROCEDURE EXTERNAL;
BEGIN;
   SALVA CONTEXTO DO PROCESSADOR;
   REGISTRA MENSAGEM INFORMATIVA NO HISTÓRICO DE EVENTOS
                                   (SERVIÇO LOG DO TESTADOR);
   RESTAURA CONTEXO DO PROCESSADOR;
END EXTERNAL;


PROCEDURE JOB;
BEGIN;
   INSTALA ESTE PROGRAMA COMO UMA IST PARA O TESTADOR
                               (SERVIÇO INSTAL DO TESTADOR);
   INSTALA A ROTINA EXTERNAL COMO HANDLER DE TRATAMENTO
                 DE EVENTO EXTERNO (SERVIÇO SETEXTEVEHDL);
   INICIALIZA O ENDEREÇO DA PORTA INFERIOR (SERVIÇO
                                                   INCPORT);
   OBTEM DO TESTADOR UMA UD VAGA (SERVIÇO RESERVE);
   LOOP;
      IF ( POSSUI UMA UD )
```

```
      THEN
         PREENCHE A UD ( O ENDEREÇO DESTINO É O
                PRÓPRIO ENDEREÇO DE NÍVEL INFERIOR DA IST);
         ENVIA A UD PELO NÍVEL INFERIOR (SERVIÇO SEND);
      END;
      RECEBE UMA UD PELA PORTA INFERIOR (SERVIÇO RECEIVE);
      GASTA TEMPO; (* LOOP DE SOFTWARE *)
  END; (* LOOP *)
END JOB;
```

Este algoritmo é bastante simples : Ele realiza apenas um laço infinito de monólogo através do nível inferior. Isto é suficiente para exercitar a maioria dos serviços do testador.

A rotina EXTERNAL implementa o "handler" de tratamento de evento externo da IST. Ela simplesmente registra mensagens informativas na lista de eventos.

Diversas ocorrências deste algoritmo, configuradas com diferentes endereços de porta inferior, foram instaladas simultaneamente para testar as varreduras de ISTs.

Os recursos da IHM foram exercitados manualmente pelo usuário.

A depuração foi realizada com o auxílio do sistema de depuração em tempo real MODULA2 (RTD - "Run Time Debugger"). O testador é compatível com o RTD, de forma que este pode ser usado para depurar "on-line" tanto o próprio testador quanto as ISTs.

IV.2 - TESTE DE UM PROTOCOLO PONTO-A-PONTO

O objetivo deste teste (e do subsequente) foi adquirir experiência na utilização do testador para o desenvolvimento de algoritmos reais e realizar uma

avaliação do mesmo.

Primeiramente adotou-se, de forma modificada, um protocolo de nível enlace de dados (" Data Link Layer "), ponto-a-ponto, apresentado por TANENBAUM [21] (cap4; sec 4.2; pag 158), denominado PROTOCOL5. Neste protocolo foram realizados testes automáticos e manuais.

Esta escolha se deve ao fato de que para a demonstração de técnicas e metodologias aplicáveis ao projeto de protocolos de comunicação, bem como para a validação de ferramentas de simulação e teste auxiliares a tais projetos, é, muitas vezes, utilizado o "one bit sliding window protocol", também chamado na literatura de "alternating bit protocol" [10]. O protocolo adotado é uma extensão deste, para tamanhos de janela maiores que um.

A modificação introduzida no PROTOCOL5 foi uma extensão, no sentido de permitir o tráfego unidirecional de dados. Para tanto foi introduzido o envio de um reconhecimento ("Ack") em separado, caso, após a recepção de um quadro, expire um "Timeout" determinado, sem que a estação receba dados para enviar no sentido oposto. O PROTOCOL5 original assume que o canal possui tráfego bidirecional intenso, de forma que os reconhecimentos são sempre em "piggyback".

O algoritmo modificado é mostrado abaixo. São adotadas as mesmas convenções e identificadores de TANENBAUM [21].

```
(* Constantes e Tipos de dados *)

CONST MAXSEQ=7;TAMMSG=15;

TYPE SEQUENCENR=(0..MAXSEQ);
     MESSAGE=ARRAY[0..TAMMSG-1]OF CHAR;
     EVTYPE=(HOSTREADY,FRAMEARRIVAL,CHKSUMERR,TIMEOUT,
             HOSTIDLE);
     FRAMEKIND=(DATA,ACK);
     FRAME=RECORD
```

```
            KIND:FRAMEKIND;
            SEQ,ACK:SEQUENCENR;
            INFO:MESSAGE;
        END;

(* Rotinas de sistema *)

PROCEDURE WAIT ( VAR EVENT : EVTYPE );
(* AGUARDA O ACONTECIMENTO DE UM EVENTO E RETORNA O SEU
   TIPO EM EVENT *)

PROCEDURE FROMHOST ( VAR M : MESSAGE );
(* BUSCA DADOS DO HOSPEDEIRO PARA TRANSMISSÃO NO CANAL *)

PROCEDURE TOHOST ( M : MESSAGE );
(* ENTREGA AO HOSPEDEIRO DADOS RECEBIDOS PELO CANAL *)

PROCEDURE GETF ( VAR R : FRAME );
(* RETORNA EM R O FRAME RECEBIDO PELO CANAL *)

PROCEDURE SENDF ( S : FRAME );
(* ENVIA PELO CANAL O FRAME S *);

PROCEDURE STARTTIMER ( K : SEQUENCENR );
(* DISPARA O TEMPORIZADOR DE TIMEOUT PARA O FRAME DE
   INDICE K *)

PROCEDURE STOPTIMER ( K : SEQUENCENR );
(* DESARMA O TEMPORIZADOR DE TIMEOUT PARA O FRAME DE
   INDICE K *)

PROCEDURE STARTACKTIMER;
(* DISPARA O TEMPORIZADOR ESPECIAL PARA O ENVIO DE
   RECONHECIMENTOS EM SEPARADO  *)

PROCEDURE STOPACKTIMER;
(* DESARMA O TEMPORIZADOR ESPECIAL PARA O ENVIO DE
   RECONHECIMENTOS EM SEPARADO  *)
PROCEDURE ENABLEHOST;
```

```
(* PERMITE AO HOSPEDEIRO GERAR EVENTOS HOSTREADY *)

PROCEDURE DISABLEHOST;
(* PROÍBE O HOSPEDEIRO DE GERAR EVENTOS HOSTREADY *)

PROCEDURE PROTOCOL5;

VAR NEXTFRAMETOSEND,ACKEXPECTED,FRAMEEXPECTED,
                                    I,NBUFFERED : SEQUENCENR;
    BUFFER : ARRAY[SEQUENCENR]OF MESSAGE;
    R,S : FRAME; EVENTO : EVETYPE;

PROCEDURE INC ( VAR K : SEQUENCENR );
BEGIN; IF K < MAXSEQ THEN K := K + 1 ; ELSE K := 0; END;
END INC;

PROCEDURE BETWEEN ( A, B, C : SEQUENCENR ) : BOOLEAN;
BEGIN;
   IF( ( A <= B ) AND ( B < C ) )
     OR
     ( ( C < A ) AND ( A <= B ) )
     OR
     ( ( C < A ) AND ( B < C ) )
   THEN RETURN TRUE; ELSE RETURN FALSE; END;
END BETWEEN;

PROCEDURE SENDDATA ( FK : FRAMEKIND; FRAMENR : SEQUENCENR);
BEGIN;
   S.KIND := FK;
   IF ( FK = DATA ) THEN
      S.INFO := BUFFER [ FRAMENR ];
      S.SEQ := FRAMENR;
   END;
   S.ACK := ( FRAMEEXPECTED + MAXSEQ ) MOD ( MAXSEQ + 1 );
   SENDF ( S );
   IF ( FK = DATA ) THEN STARTTIMER ( FRAMENR );
   STOPACKTIMER;
END SENDDATA;
BEGIN; (* PROTOCOL5 *)
```

```
   ENABLEHOST;NEXTFRAMETOSEND:=0;ACKEXPECTED:=0;
   FRAMEEXPECTED:=0;NBUFFERED:=0;
   LOOP
      WAIT ( EVENT );
      CASE EVENT OF
         HOSTREADY :
            FROMHOST ( BUFFER [ NEXTFRAMETOSEND ] );
            NBUFFERED := NBUFFERED + 1;
            SENDDATA ( DATA , NEXTFRAMETOSEND );
            INC ( NEXTFRAMETOSEND ) |
         FRAMEARRIVAL :
            GETF ( R );
            IF ( R.KIND = DATA ) THEN
               IF ( R.SEQ = FRAMEEXPECTED ) THEN
                  TOHOST ( R.INFO );
                  INC ( FRAMEEXPECTED );
                  STARTACKTIMER;
               END;
            END;
            WHILE BETWEEN ( ACKEXPECTED , R.ACK ,
                                       NEXTFRAMETOSEND) DO;
               NBUFFERED := NBUFFERED - 1 ;
               STOPTIMER ( ACKEXPECTED );
               INC ( ACKEXPECTED );
            END |
         CHKSUMERR : |
         TIMEOUT :
            NEXTFRAMETOSEND := ACKEXPECTED;
            FOR I:=1 TO NBUFFERED DO
               SENDDATA ( DATA , NEXTFRAMETOSEND );
               INC ( NEXTFRAMETOSEND );
            END |
         HOSTIDLE : SENDDATA ( ACK , 0 );
      END; (* CASE *)
      IF ( NBUFFERED < MAXSEQ )
      THEN ENABLEHOST;
      ELSE DISABELHOST; END;
   END;
END PROTOCOL5;
```

IV.2.1 - IMPLEMENTAÇÃO

O programa principal, cujo algoritmo é mostrado no item anterior, é implementado pelo processo PROTOCOL5. O trabalho de implementação compreendeu ainda a codificação das primitivas de biblioteca do sistema e a realização do procedimento de instalação e configuração das ISTs.

As primitivas FROMHOST, TOHOST, ENABLEHOST, DISABLEHOST, GETF e SENDF fazem parte do módulo INTERFACE. STOPTIMER, STARTTIMER, STARTACKTIMER e STOPACKTIMER pertencem ao módulo TIMER. WAIT foi implementada com o uso das primitivas do NUCLEUS. A implementação das primitivas é descrita nos próximos itens.

O procedimento de instalação da IST evita que o seu NUCLEUS modifique o vetor de interrupções da UCP, o que a tornaria incompatível com o testador. A rotina de tratamento de relógio do NUCLEUS é configurada para o testador como rotina de evento externo da IST.

IV.2.1.1 - EVENTOS

O mecanismo de sinalização de eventos é implementado por meio de troca de mensagens usando as primitivas do NUCLEUS. As mensagens de eventos têm o seguinte formato :

```
EVEMSGPTR=POINTER TO EVEMSG;
EVEMSG=RECORD
          XXX:ADDRESS;
          EVENT:EVTYPE;
          PARAM:SEQUENCENR;
       END;
```

Onde :

. XXX - Campo para uso do NUCLEUS.

. EVENT - Tipo do evento. Os tipos válidos estão no "set" EVTYPE e são :

a) FRAMEARRIVAL - Recepção de quadro pelo nível inferior;

b) CHKSUMERR - Recepção de quadro inválido pelo nível inferior;

c) TIMEOUT - Expirou o tempo de espera pelo reconhecimento do quadro cujo índice está indicado em PARAM;

d) HOSTREADY - Recepção de uma mensagem pelo nível superior;

e) HOSTIDLE - Expirou o tempo de espera pela chegada de dados a enviar a fim de fazer por "piggyback" o reconhecimento dos quadros pendentes.

. PARAM - Usado no evento TIMEOUT para indicar o índice do quadro cujo contador expirou.

As mensagens de eventos existem, em número de 20, em um saco ("pool") que é a caixa-postal POOLEVEMBX. A sinalização de um evento consiste simplesmente em obter nesta caixa-postal uma mensagem vaga, preenchê-la e enviá-la para a caixa postal EVEMBX, onde o processo PROTOCOL5 permanece em espera bloqueada aguardando a ocorrência de eventos. Após tratar o evento, PROTOCOL5 devolve a mensagem de evento ao saco.

Os eventos FRAMEARRIVAL e HOSTREADY são gerados pelo módulo INTERFACE. TIMEOUT e HOSTIDLE são gerados pelo TIMER. CHKSUMERR não é gerado, pois pode ser simulado pelo usuário na IHM (descartando quadros).

IV.2.1.2 - TIMER

O módulo TIMER provê os serviços de temporização solicitados pelo PROTOCOL5. Os temporizadores são implementados, segundo a sugestão de TANENBAUM [21] (cap4; sec 4.2; pag 157), por meio de uma fila de contadores simplesmente encadeada. Esta fila tem a estrutura :

```
TMTYPEPTR = POINTER TO TMTYPE;
TMTYPE = RECORD
            NEXT : TMTYPEPTR;
            TIME : CARDINAL;
            INDEX : SEQUENCENR;
         END;
```

Onde :

. NEXT - Apontador para o próximo contador;

. TIME - Valor atual do contador. Este valor é inicializado com o tempo (em unidades de 54msec) a ser esperado, e é decrementado, a cada ciclo de relógio, até atingir zero. Ao expirar, o contador é retirado da fila de contadores e o evento correspondente é gerado.

. INDEX - Indice do quadro a que corresponde o contador.

Os contadores existem, em número de 20, em um saco, de onde devem ser obtidos a priori e para onde devem ser devolvidos a posteriori do uso.

O temporizador especial para o reconhecimento em separado é implementado por um contador específico (ACKTIMER).

Os temporizadores, exceto o ACKTIMER, são "retrigáveis".

O "timeout" para retransmissão de um pacote foi arbitrariamente fixado em 432msec (8 ciclos de relógio) e o "Timeout" para o reconhecimento em separado em 270msec

(5 ciclos de relógio).

O gerenciamento de todos os temporizadores é feito pelo processo TIMERTASK, que executa a cada ciclo de relógio. Esta frequência é obtida por meio de espera temporizada (um ciclo de relógio) em uma caixa-postal para a qual não são enviadas mensagens. O algoritmo de TIMERTASK é mostrado abaixo.

A mútua exclusão para o acesso (TIMERTASK e rotinas) às variáveis globais do TIMER (fila de contadores, ACKTIMER e saco) é obtida através da caixa-postal TMMUTXMBX.

```
PROCEDURE TIMERTASK;
BEGIN;
   LOOP
      WAIT ( UM TICK DE RELÓGIO );
      WAIT ( MUTUA EXCLUSÃO TIMER );
      IF ( EXISTE ALGUM TEMPORIZADOR ARMADO ) THEN
         DECREMENTA O CONTADOR DO TOPO DA FILA;
         IF ( O CONTADOR DO TOPO EXPIROU ) THEN
            LOOP
               RETIRA O CONTADOR DO TOPO DA FILA;
               DEVOLVE O CONTADOR AO SACO DE CONTADORES;
               OBTEM UMA MENSAGEM DE EVENTO NO SACO
                                              CORRESPONDENTE;
               SEND ( EVENTO TIMEOUT );
               IF ( O PRÓXIMO CONTADOR NÃO ESTA EXPIRADO )
               THEN EXIT;
            END; (* LOOP *)
         END;
      END;
      IF ( ACKTIMER NÃO ESTÁ EXPIRADO ) THEN
         DECREMENTA ACKTIMER;
         IF ( ACKTIMER EXPIROU ) THEN
            OBTEM UMA MENSAGEM DE EVENTO NO SACO;
            SEND ( EVENTO HOSTIDLE );
         END;
      END;
```

```
      SEND ( MUTUA EXCLUSÃO TIMER );
   END;
END TIMERTASK;
```

Os algoritmos das rotinas do TIMER são apresentados abaixo:

```
PROCEDURE DISARMTIMER ( IND : SEQUENCENR );
BEGIN;
   IF ( A LISTA DE CONTADORES NÃO ESTÁ VAZIA )
      AND
      ( O PRIMEIRO CONTADOR NÃO TEM ÍNDICE IND )
   THEN
      APONTA PARA O INÍCIO DA LISTA DE CONTADORES;
     LOOP
         IF ( CHEGOU A FIM DA LISTA DE CONTADORES )
         THEN EXIT;
         ELSE
            IF ( O PRÓXIMO CONTADOR NÃO TEM INDICE IND )
            THEN APONTA PARA O PRÓXIMO CONTADOR;
            ELSE
               RETIRA DA LISTA O PRÓXIMO CONTADOR;
               CORRIGE O VALOR DO CONTADOR SEGUINTE,
                   INCREMENTANDO-O DO VALOR DO
                             CONTADOR RETIRADO;
                   DEVOLVE O CONTADOR RETIRADO AO SACO DE
                CONTADORES;
               EXIT;
            END;
         END;
      END; (* LOOP *)
   ELSE
      IF ( A LISTA NãO ESTÁ VAZIA ) THEN
         RETIRA O PRIMEIRO CONTADOR DA LISTA;
         DEVOLVE AO SACO O CONTADOR RETIRADO;
      END;
   END;
END DISARMTIMER;
PROCEDURE STARTTIMER ( IND : SEQUENCENR );
```

```
BEGIN;
   WAIT ( MUTUA EXCLUSÃO TIMER );
   DISARMTIMER ( IND );
   IF ( EXISTE CONTADOR VAGO NO SACO DE CONTADORES ) THEN
      OBTEM DO SACO UM CONTADOR VAGO;
      PREENCHE O NOVO CONTADOR ( O VALOR DO CONTADOR É
                                 INICIALIZADO COM O TIMEOUT);
      IF ( A LISTA DE CONTADORES NÃO ESTÁ VAZIA )
         AND
         ( O PRIMEIRO CONTADOR TEM VALOR ATUAL MENOR QUE
                                    O VALOR DO NOVO CONTADOR)
      THEN
         APONTA PARA O INÍCIO DA LISTA DE CONTADORES;
         LOOP
            IF (CHEGOU AO FIM DA LISTA DE CONTADORES) THEN
               DECREMENTA O VALOR DO NOVO CONTADOR
                              DO VALOR DO ÚLTIMO CONTADOR;
               INSERE O NOVO CONTADOR NO FIM DA LISTA DE
               CONTADORES ;
               EXIT;
            ELSE
               IF ( O VALOR ATUAL DO PRÓXIMO CONTADOR É
                    MENOR QUE O VALOR DO NOVO CONTADOR) THEN
                  DECREMENTA O VALOR DO PRÓXIMO CONTADOR
                   DO VALOR DO NOVO CONTADOR;
                  APONTA PARA O PRÓXIMO CONTADOR;
               ELSE
                  DECREMENTA O VALOR DO NOVO CONTADOR
                        DO VALOR DO PRÓXIMO CONTADOR;
                  INSERE O NOVO CONTADOR ANTES DO PRÓXIMO;
                  EXIT;
               END;
            END;
         END; (* LOOP *)
      ELSE
         IF ( A LISTA NãO ESTÁ VAZIA )
         THEN DECREMENTA O VALOR DO PRIMEIRO CONTADOR
                                    DO VALOR DO NOVO CONTADOR;
         INSERE O CONTADOR NO INÍCIO DA FILA DE CONTADORES;
```

```
      END;
   END;
   SEND ( MUTUA EXCLUSÃO TIMER );
END;


PROCEDURE STOPTIMER ( IND : SEQUENCENR );
BEGIN;
   WAIT ( MUTUA EXCLUSÃO TIMER );
   DISARMTIMER ( IND );
   SEND ( MUTUA EXCLUSÃO TIMER );
END STOPTIMER;


PROCEDURE STARTACKTIMER
BEGIN;
   WAIT ( MUTUA EXCLUSÃO TIMER );
   IF ( ACKTIMER ESTA EXPIRADO ) THEN INICIALIZA ACKTIMER;
   SEND ( MUTUA EXCLUSÃO TIMER );
END STARTACKTIMER;


PROCEDURE STOPACKTIMER;
BEGIN;
   WAIT ( MUTUA EXCLUSÃO TIMER );
   RESETA ACKTIMER;
   SEND ( MUTUA EXCLUSÃO TIMER );
END STOPACKTIMER;
```

IV.2.1.3 - INTERFACE

O módulo INTERFACE provê a interface do PROTOCOL5 com o testador.

A recepção de dados é realizada por dois processos :

. NITASK - nivel inferior;
. HSTASK - nível superior.

Eles executam a cada ciclo de relógio (esta frequência é obtida por meio de espera temporizada em uma caixa-postal para a qual não são enviadas mensagens) e utilizam os serviços do testador para esvaziar as filas de entrada da IST. Os dados (mensagens - nível superior; quadros - nível inferior) contidos nas UDs recebidas são armazenados nas filas internas do módulo e os eventos correspondentes são gerados.

Existe uma fila para cada nível. O acesso a elas é feito através das primitivas GETF (nível inferior) e FROMHOST (nível superior), que retornam os dados contidos na cabeça da fila correspondente.

O envio de dados é feito por TOHOST (nível superior) e SENDF (nível inferior), que acessam diretamente os serviços do testador.

Abaixo são mostrados os formatos das filas internas do INTERFACE.

```
(* Nível Inferior *)
NIMSGPTR : POINTER TO NIMSGTYPE;
NIMSGTYPE : RECORD
               NEXT : NIMSGPTR;
               FR : FRAME;
            END;
```

Onde :

. NEXT - Apontador para o próximo elemento;

. FR - Conteúdo de quadro recebido.

```
(* Nível Superior *)
HSMSGPTR : POINTER TO HSMSGTYPE;
HSMSGTYPE : RECORD
               NEXT : HSMSGPTR;
```

```
        MS : MESSAGE;
    END;
```

Onde :

. NEXT - Apontador para o próximo elemento;

. MS - Conteúdo de mensagem recebida.

Os "buffers" dos tipos NIMSGTYPE e HSMSGTYPE são disponíveis em sacos do INTERFACE, de forma a possibilitar o enfileiramento de até 20 mensagens de cada tipo.

A habilitação/desabilitação do hospedeiro é obtida através de uma variável global do módulo, modificada pelas rotinas ENABLEHOST e DISABLEHOST :

```
HSSTATUSSET : ( ENABLED , DISABLED );
HSSTAUTS : HSSTATUSSET;
```

O tratamento do hospedeiro (ENABLEHOST E DISABLEHOST) é feito de forma simplificada, pois assume-se que o tamanho da janela é compatível com a velocidade do canal e com a sua taxa de utilização pelo hospedeiro. O tratamento completo de DISABLEHOST implica em percorrer a fila de eventos pendentes, retirando todos os eventos "HOSTREADY" ali contidos. Estes devem ser armazenados (com a sua ordem mantida) para posterior retorno, feito por ENABLEHOST.

A mútua exclusão nas variáveis globais é obtida através de duas caixas-postais : HSMUTXMBX e NIMUTXMBX, para as variáveis relacionadas ao nível superior e inferior, respectivamente.

O algoritmo do processo HSTASK é apresentado abaixo. O NITASK é o análogo para o nível inferior.

```
(* Nível superior *)
```

```
PROCEDURE HSTASK;
BEGIN;
   LOOP
      WAIT ( UM CICLO DE RELÓGIO );
      WAIT ( MÚTUA EXCLUSÃO NÍVEL SUPERIOR );
      IF ( HOST ESTÁ HABILITADO ) THEN
         LOOP
            OBTÉM UD DA FILA DE ENTRADA DE NÍVEL
                    SUPERIOR DA IST ( SERVIÇO RECEIVE DO
                                                TESTADOR);
            IF ( NÃO RECEBEU UD ) THEN EXIT;
            ELSE
               OBTEM ELEMENTO HSMSGTYPE DO SACO
                                          CORRESPONDENTE;
               PREENCHE ELEMENTO COM O CONTEÚDO DA
                                             UD RECEBIDA;
               INSERE ELEMENTO NA FILA INTERNA DE
                   MENSAGENS RECEBIDAS PELO NÍVEL SUPERIOR;
               DEVOLVE A UD RECEBIDA ( SERVIÇO RELEASE );
               OBTEM MENSAGEM VAGA NO SACO DE MENSAGENS
                                             DE EVENTOS;
               SEND ( EVENTO HOST READY );
            END;
         END; (* LOOP *)
      END;
      SEND ( MÚTUA EXCLUSÃO NÍVEL SUPERIOR );
   END; (* LOOP *)
END HSTASK;
```

Os algoritmos das rotinas do INTERFACE são mostrados abaixo. As rotinas GETF e SENDF são análogas a FROMHOST e TOHOST.

```
PROCEDURE ENABLEHOST;
BEGIN;
   WAIT ( MÚTUA EXCLUSÃO NÍVEL SUPERIOR );
   HSSTATUS := ENABLED;
   SEND ( MÚTUA EXCLUSÃO NÍVEL SUPERIOR );
```

```
END ENABLEHOST;

PROCEDURE DISABLEHOST;
BEGIN;
   WAIT ( MÚTUA EXCLUSÃO NÍVEL SUPERIOR );
   HSSTATUS := DISABLED;
   SEND ( MÚTUA EXCLUSÃO NÍVEL SUPERIOR );
END DISABLEHOST;

PROCEDURE FROMHOST ( VAR MSGDATA : MESSAGE );
BEGIN;
   WAIT ( MÚTUA EXCLUSÃO NÍVEL SUPERIOR );
   RETIRA A CABEÇA DA FILA INTERNA DE
                                MENSAGENS DO NÍVEL SUPERIOR;
   MSGDATA RECEBE O CONTEÚDO DA MENSAGEM;
   DEVOLVE A MENSAGEM AO SACO INTERNO DE MENSAGENS
                                          DO NÍVEL SUPERIOR;
   SEND ( MÚTUA EXCLUSÃO NÍVEL SUPERIOR );
END FROMHOST;

PROCEDURE TOHOST ( VAR MSGDATA : MESSAGE );
BEGIN;
   OBTEM UMA UD VAGA DO TESTADOR ( SERVIÇO RESERVE
                                                DO TESTADOR);
   PREENCHE A UD COM O CONTEÚDO DE MSGDATA;
   ENVIA A UD PARA A FILA DE SAÍDA SUPERIOR
                                       DA IST (SERVIÇO SEND);
END TOHOST;
```

IV.2.2 - TESTE MANUAL

Este teste foi feito em duas etapas : primeiramente exercitando-se apenas uma IST, para detectar os erros de mais baixo nível e os mais evidentes; e a seguir com o sistema completo, composto por duas ISTs idênticas (diferentes apenas pelos endereços para o testador) que se comunicam.

As entradas manuais utilizadas para exercitar as ISTs

foram:

. Quadros (UDs de nível inferior);

. Reconhecimentos em separado (UDs de nível inferior);

. Mensagens (UDs de nível superior);

. Descartar quadros.

O procedimento de condução manual do teste consistiu em editar UDs correspondentes a estes estímulos, enviá-las para as ISTs e analisar as saídas geradas em resposta a eles. Para isto, foram utilizados praticamente todos os recursos oferecidos pela IHM do testador. As sequências de entradas foram geradas intuitivamente pelo usuário.

Os testes foram ainda auxiliados pelo sistema de depuração MODULA2.

Apesar da simplicidade do PROTOCOL5, foram cometidos diversos erros de implementação e alguns de concepção, listados abaixo. A maior parte destes erros estava localizada nos módulos INTERFACE e TIMER.

Os erros no INTERFACE eram previsíveis, em virtude do processamento de baixo nível por ele realizado. Os erros de concepção do TIMER (um deles foi detectado por este teste manual e o outro por meio do teste automático, como veremos no próximo item) são devidos ao fato de que TANENBAUM [21] não especificou o gerenciamento das temporizações.

No decorrer dos testes foram detectados alguns erros no testador. Assim eles serviram ainda como uma etapa de depuração final do mesmo.

A coletânea de erros detectados no PROTOCOL5, apresentada abaixo, visa mostrar os tipos de erros que podem ser

A coletânea de erros detectados no PROTOCOL5, apresentada abaixo, visa mostrar os tipos de erros que podem ser rastreados em testes manuais.

No item EFEITO são relatados os "sintomas" do erro, que foram detectados com o auxílio do testador. No item CAUSA são apontados os erros de implementação ou projeto que conduziram a estas falhas. Os relatos são feitos de forma sucinta.

a)
. EFEITO
À recepção de qualquer quadro (nível inferior) a IST (e o restante do sistema em teste) bloqueava e recaía em "range error" (erro do RTS MODULA2);
. CAUSA
Módulo INTERFACE. Erro em limites de comando iterativo (comando "FOR") na rotina TOHOST.

b)
. EFEITO
À recepção de qualquer mensagem (nível superior) a IST recaía em "range error";
. CAUSA
Módulo INTERFACE. Erro em limites de comando iterativo na rotina SENDF.

c)
. EFEITO
Bloqueio da IST à recepção de um quadro ou mensagem. A IST não realizava mais nenhum processamento;
. CAUSA
Módulo TIMER. Interbloqueio, TIMERTASK não devolvia a mútua exclusão do TIMER. Isto causava o bloqueio do processo PROTOCOL5.

d)
. EFEITO
Os quadros enviados pela IST continham campos de controle inválidos. As UDs correspondentes também apresentavam erros no seu preenchimento;
. CAUSA
Processo PROTOCOL5 e Módulo INTERFACE. Erros no preenchimento dos quadros em SENDDATA e no preenchimento das UDs de nível inferior em SENDF.

e)
. EFEITO
Erro na passagem ao hospedeiro dos dados contidos nos quadros recebidos. Eram passados menos octetos que os recebidos;
. CAUSA
Erro do usuário e no Módulo INTERFACE. Erro no preenchimento dos quadros na IHM e nos limites de comando iterativo em TOHOST.

f)
. EFEITO
À segunda recepção de quadro, os dados contidos no quadro recebido não eram passados corretamente ao hospedeiro;
. CAUSA
Módulo INTERFACE. Erros (rotina GETF) na manipulação de apontadores da fila interna de quadros recebidos.

g)
. EFEITO - À segunda recepção de mensagem pelo nível superior, o quadro correspondente, enviado através do nível inferior, não era preenchido corretamente com os dados;
. CAUSA
Módulo INTERFACE. Erros (rotina FROMHOST) na manipulação de apontadores da fila interna de mensagens recebidas.

h)

. EFEITO

Após receber várias mensagens e/ou quadros e tratá-los corretamente a IST ficava bloqueada;

. CAUSA

Processo PROTOCOL5. Interbloqueio, PROTOCOL5 não devolvia corretamente ao saco as mensagens de eventos. Estas esgotavam-se e os processos HSTASK e NITASK bloqueavam à sua espera.

i)

. EFEITO

Em caso de "timeouts" sucessivos de mais de um quadro o número de retransmissões aumentava exponencialmente. Supondo dois quadros pendentes : no primeiro "timeout" se verificavam duas retransmissões - uma para cada quadro; no segundo "timeout" se verificavam quatro retransmissões - duas para cada quadro; no terceiro "timeout" oito retransmissões - quatro por quadro; etc. Isto terminava por bloquear o sistema com "system halted" (erro do RTS);

. CAUSA

Módulo TIMER. Erro de concepção no tratamento dos temporizadores, que devem ser "retrigáveis". Os temporizadores dos quadros não eram "retrigados" por STARTTIMER e sim re-inseridos. Isto causava o aumento do número de "timeouts" e consequentemente o de retransmissões. Os temporizadores disponíveis acabavam esgotando-se o que recaía no "system halted".

IV.2.3 - TESTE AUTOMÁTICO

O teste automático realizado no PROTOCOL5 foi do tipo teste aleatório. Com isto, evitamos recair no problema da geração de sequências de teste, pois este foge ao escopo deste trabalho.

Para a realização deste teste, é necessária a síntese dos observadores correspondentes. Neste caso, eles foram

sintetizados a partir de um modelo formal do protocolo, realizado através de Redes de Petri.

O modelo completo do PROTOCOL5 em Redes de Petri encontra-se no APÊNDICE A. Este modelo inclui, não somente o algoritmo do protocolo, mas também os algoritmos das rotinas de sistema envolvidas (o modelo admite o tratamento simplificado do hospedeiro, a exemplo do algoritmo). O modelo do APÊNDICE A corrresponde ao PROTOCOL5 com tamanho de janela unitário. Este modelo é extensível para tamanhos de janela maiores .

A partir do modelo completo obtivemos empiricamente, por meio de simplificações, os observadores locais (OLI e OLS). Os modelos destes observadores estão na figura IV-1 (As convenções nela adotadas são apresentadas no APÊNDICE A).

Para o OLI as simplificações adotadas foram as seguintes :

. Extração de toda informação referente à temporização contida no modelo completo. O teste deste tipo de assertiva foi considerado difícil no testador;

. Extração da parte relacionada às rotinas do sistema.

Com isto, obtivemos o modelo da figura IV-1a ,que descreve o comportamento do protocolo com relação às UDs de seu nível inferior. Este modelo representa as relações entre os campos de controle (SEQ, ACK, KIND) das sequências de quadros que circulam através desta interface (e em relação às suas variáveis internas), e pode ser visto como uma especificação funcional.

O modelo da figura IV-1a também assume tamanho unitário de janela, mas pode ser estendido para tamanhos maiores.

Para o OLS (figura IV-1b), o modelo descrevendo o serviço do nível superior é elementar e resume a assertiva : "Todos

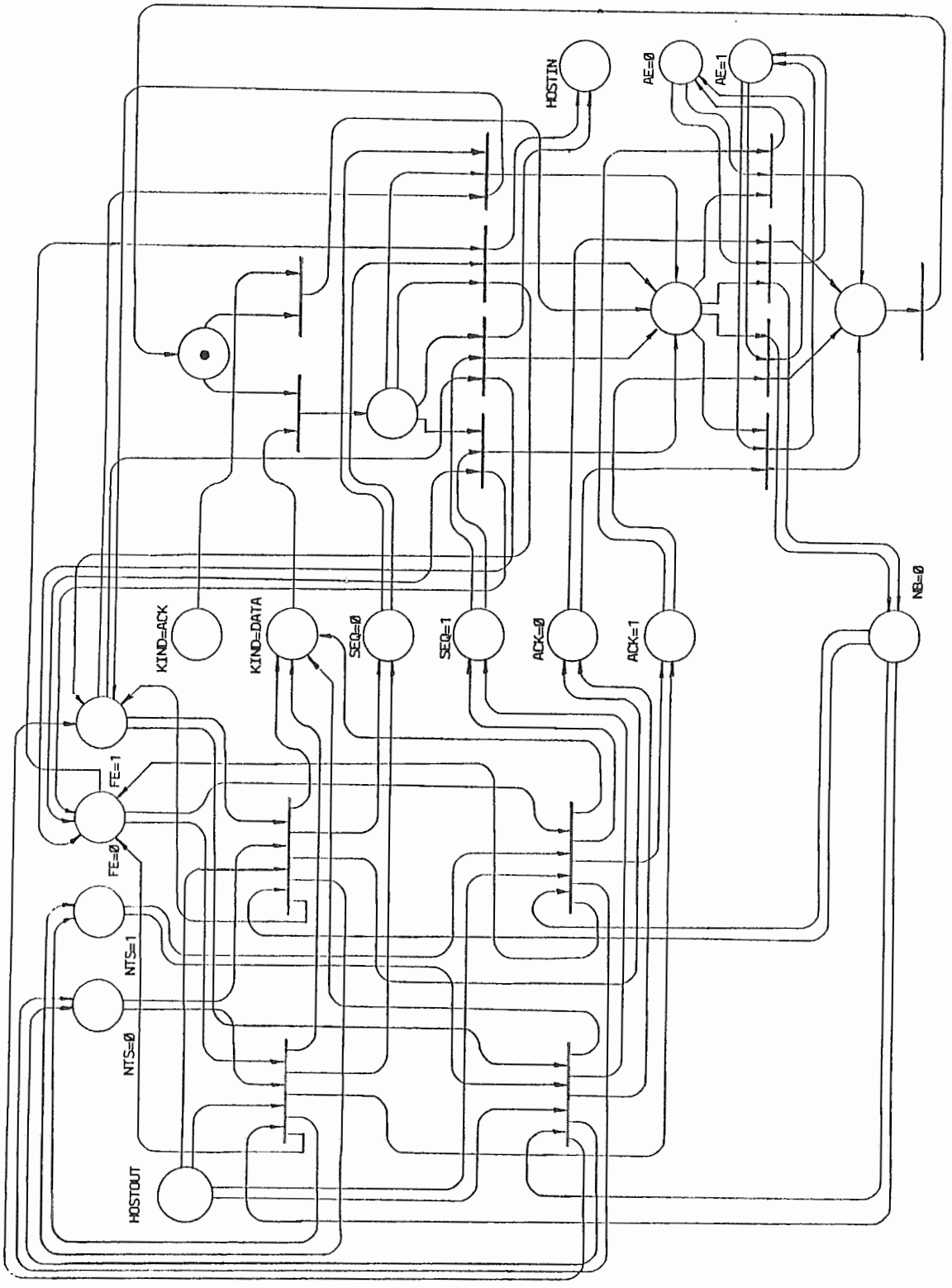
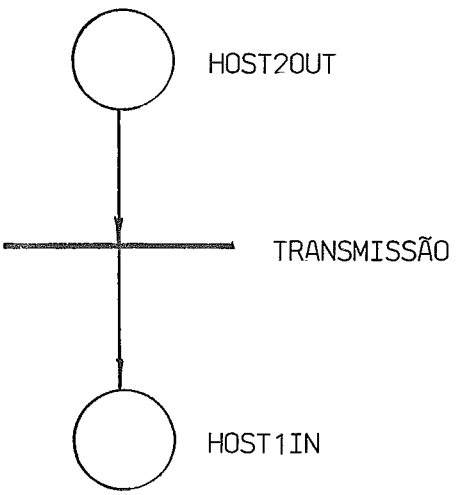


Figura IV-1(a) - Observador Local Inferior para o PROTOCOL5



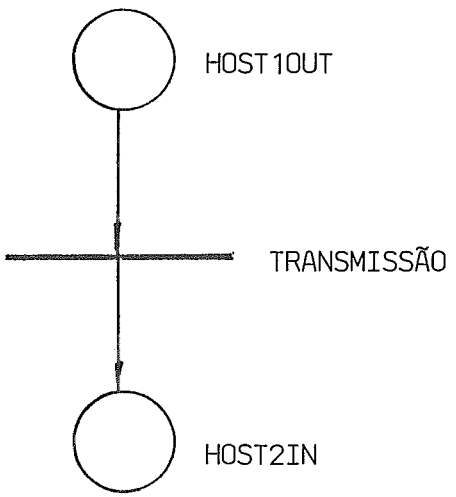


Figura IV-1(b) - Observador Local Superior para o PROTOCOL5

os dados enviados por um hospedeiro devem ser recebidos pelo outro hospedeiro na ordem correta". A simplificação é imediata, basta abstrair de todo o modelo do protocolo.

IV.2.3.1 - IMPLEMENTAÇÃO DOS OBSERVADORES

. OLI

O OLI testa o nível inferior através das relações entre os campos de controle das sequências de quadros que circulam através deste. Se uma condição inválida é detectada o OLI insere a mensagem informativa correspondente na lista de eventos.

As mensagens geradas são :

I) "OLI - RETRANSMISSÃO OU FRAME INVÁLIDO"
Se o campo SEQ de algum quadro é diferente do esperado. Isto indica quadro fora de sequência.

II) "OLI - ACK INVÁLIDO"
Se o campo ACK de algum quadro é diferente do esperado. Isto indica que o quadro que está sendo reconhecido é diferente do último recebido.

O algoritmo do OLI para uma ISTn genérica é mostrado abaixo. Se quisermos testar as duas ISTs é necessário repetir o código para cada uma delas.

```
PROCEDURE OLI;
BEGIN;
   WHILE ( EXISTEM UDs EM Q8 ) DO
      RETIRA UD DE Q8;
      IF ( A UD FOI TRANSMITIDA PELA ISTn ) THEN
         IF ( A UD CONTEM UM QUADRO DE DADOS ) THEN
            IF ( O NÚMERO DE SEQUÊNCIA DO QUADRO É
                  DIFERENTE DO PRÓXIMO QUADRO A TRANSMITIR)
            THEN REGISTRA MENSAGEM " RETRANSMISSÃO OU
```

```
                                                FRAME INVÁLIDO ";
            ELSE INCREMENTA O NUMERO DE SEQUÊNCIA
                    DO PRÓXIMO QUADRO A TRANSMITIR; END;
         END;
         IF ( O NÚMERO DO RECONHECIMENTO DO QUADRO É
                 DIFERENTE DO ÚLTIMO QUADRO RECEBIDO ) THEN
           REGISTRA MENSAGEM " ACK INVÁLIDO ";
         END;
      END;
      INSERE UD EM Q6;
   END; (* WHILE *)
   WHILE ( EXISTEM UDs EM Q7 ) DO
      RETIRA UD DE Q7;
      IF ( A UD É DESTINADA À ISTn ) THEN
         IF ( A UD CONTEM UM QUADRO DE DADOS ) THEN
            IF ( O NÚMERO DE SEQUÊNCIA DO QUADRO
                                RECEBIDO É O ESPERADO ) THEN
               INCREMENTA O NÚMERO DE SEQUÊNCIA DO
                                            QUADRO ESPERADO;
            END;
         END;
         IF ( O NÚMERO DO RECONHECIMENTO DO
                        QUADRO RECEBIDO É O ESPERADO ) THEN
            INCREMENTA O NÚMERO DO RECONHECIMENTO ESPERADO;
         END;
      END;
      INSERE UD EM Q9;
   END; (* WHILE *)
END OLI;
```

. OLS

A implementação do OLS é ilustrada na figura IV-2.

Todas as mensagens recebidas pelas ISTs de seus hospedeiros são armazenadas em filas do OLS, sendo uma fila para cada IST. Quando uma mensagem é passada ao hospedeiro por determinada IST, ela é comparada com a mensagem da cabeça

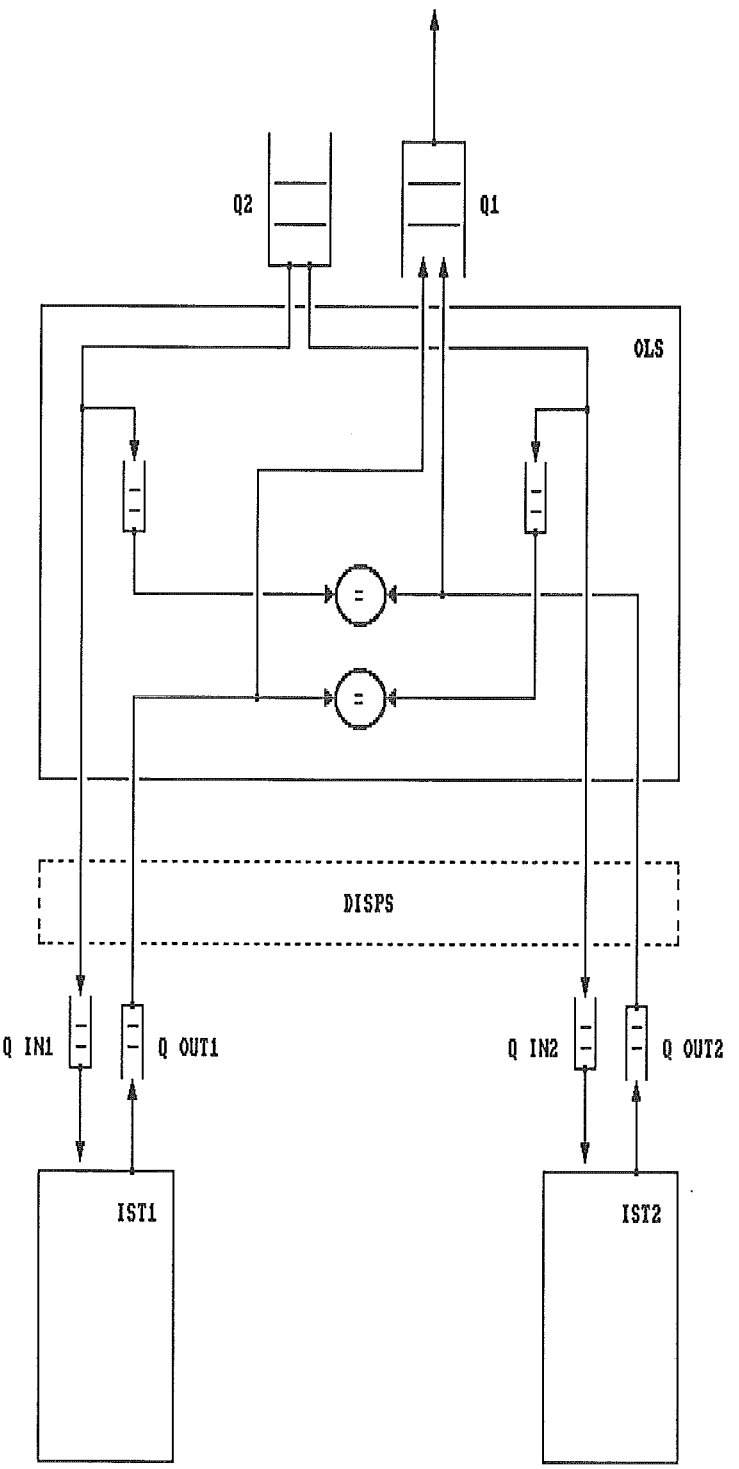


Figura IV-2 - Implementacao do OLS para o Protocol5

da fila da outra IST. Se foram iguais assume-se que a comunicação foi concluída com sucesso e a mensagem da cabeça da fila é descartada, caso contrário o OLS registra na lista de eventos o erro correspondente("OLS - ERRO NA TRANSFERÊNCIA DE DADOS AO HOST"), e a mensagem da cabeça da fila é mantida.

O OLS gera ainda um erro em caso de esgotamento de sua fila interna ("OLI - ESGOTAMENTO DE ÁREA DE TRABALHO"), que é implementada através de uma lista sequencial circular de dez elementos. Neste caso a mensagem mais antiga é perdida, pois é suplantada pela mais recente.

O algoritmo do OLS para uma ISTn genérica é mostrado abaixo. Este código deve ser repetido para as duas ISTs.

```
PROCEDURE OLS
BEGIN;
   WHILE ( EXISTEM UDs EM Q12) DO
      RETIRA UD DE Q12;
      IF ( A UD É DESTINADA À ISTn ) THEN
         IF ( A FILA INTERNA CORRESPONDENTE
                                 À ISTn NÃO ESTÁ ESGOTADA )

         THEN INSERE NO FIM DA FILA A MENSAGEM
                                              CONTIDA NA UD;
         ELSE REGISTRA MENSAGEM "OLS - ESGOTAMENTO DE
                                       ÁREA DE TRABALHO";
              INSERE NO FIM DA FILA A MENSAGEM CONTIDA
                NA UD, SUPLANTANDO A MENSAGEM MAIS ANTIGA.
         END;
      END;
      INSERE UD EM Q10;
   END; (* WHILE *)
   WHILE ( EXISTEM UDs EM Q11 ) DO
      RETIRA UD DE Q10;
      IF ( A UD CONTÉM UMA MENSAGEM QUE FOI TRANSMITIDA
           PELA OUTRA IST AO SEU HOSPEDEIRO ) THEN
         IF ( A FILA DE MENSAGENS DO OLS CORRESPONNDENTE À
```

```
                ISTn NÃO ESTÁ VAZIA ) THEN
              IF ( CONTEÚDO DA MENSAGEM NÃO COINCIDE
                                    COM A CABEÇA DA FILA)
              THEN REGISTRA "ERRO NA TRANFERÊNCIA DE
                                    DADOS AO HOST";
              ELSE LIBERA A PRIMEIRA MENSAGEM DA FILA
                                    DA ISTn; END;
           ELSE REGISTRA "ERRO NA TRANSFERÊNCIA DE DADOS AO
                                    HOST"; END;
       END;
       INSERE UD EM Q1;
   END; (* WHILE *)
END OLS;
```

. CT

O CT utilizado neste teste simula o comportamento dos hospedeiros das duas ISTs. Ele envia para o nível superior de cada uma das ISTs mensagens geradas aleatoriamente em intervalos independentes, de 1 a 12 ciclos de relógio.

O CT descarta as mensagens enviadas pelas ISTs aos seus hospedeiros.

O algoritmo do CT para uma ISTn é mostrado abaixo. Este código deve ser repetido para as duas ISTs.

```
PROCEDURE CT;
BEGIN;
   INCREMENTA CONTADOR DA ISTn;
   IF (O CONTADOR ATINGIU O NUM. DE CICLOS A ESPERAR) THEN
      GERA ALEATÓRIAMENTE NOVO NÚMERO DE
           CICLOS A ESPERAR NO INTERVALO [1..12];
      ENVIA UD ( INSERE UD EM Q3 ) CONTENDO MENSAGEM
           DESTINADA AO NÍVEL  SUPERIOR DA ISTn;
   END;
   WHILE ( EXISTEM UDs EM Q2 ) DO
      RETIRA UD DE Q2;
```

```
      IF ( A UD É DO NÍVEL SUPERIOR )
      THEN  DESCARTA A UD;
      ELSE INSERE A UD EM Q3; END;
   END; (* WHILE *)
END CT;
```

. OG e NI

Neste teste não foi utilizado observador global. O serviço do nível inferior, que consiste em gerar aleatoriamente as exceções possíveis na comunicação, também não foi implementado.

IV.2.3.2 - RESULTADOS DO TESTE AUTOMÁTICO

Neste teste foram utilizadas as duas ISTs, e o testador foi executado em varreduras contínuas.

Constatou-se que as condições de erro testadas pelo OLI ocorriam esporadicamente. Os motivos foram levantados :

. RETRANSMISSÃO OU FRAME INVÁLIDO
Era causado por um erro de concepção do modulo TIMER : O temporizador para envio do reconhecimento em separado era, erradamente, "retrigável" (rotina STARTACKTIMER).
Assim, se uma IST recebia pelo canal uma sequência de quadros em intervalos menores que o "timeout" do reconhecimento em separado, então este reconhecimento só era enviado após o último quadro. Isto causava retransmissões de quadros por parte da outra IST e, portanto, a ocorrência de quadros fora de sequência. A correção do módulo TIMER eliminou este erro.
É conveniente distinguir no observador a retransmissão de quadros da ocorrência real de quadros inválidos. Para isto basta levar em conta que as retransmissões são caracterizadas por quadros cujo número de sequência está no intervalo entre o último quadro cujo reconhecimento foi

recebido pela IST e o último quadro enviado por ela.

. ACK INVÁLIDO
Esta condição é detectada errôneamente pelo OLI.
Em caso de uma IST receber sucessivamente dois quadros e enviar um reconhecimento (em "piggyback" ou em separado) apenas do primeiro deles, o OLI acusa "ACK INVÁLIDO" pois ambos já haviam sido observados e o OLI espera o reconhecimento do último.
Isto ocorre porque os eventos são enfileirados internamente nas ISTs e pode haver o intercalamento, entre os eventos de recepção dos dois frames sucessivos, de um evento ("HOSTIDLE" ou "HOSTREADY") que cause a transmissão de um reconhecimento .
Existem duas abordagens para a eliminação deste erro :
. Realizar o teste do "ack" a partir de seus requisitos de temporização. Isto é considerado pouco prático no testador proposto, em virtude da estratégia de execução dos observadores;
. Tornar o observador mais benevolente, testando a validade do reconhecimento a partir de uma janela, cujo limite inferior é o indice do último quadro reconhecido pela IST, e cujo limite superior é o índice do próximo quadro esperado por ela . Esta é a solução mais conveniente.

As inconveniências no funcionamento do OLI são devidas ao fato de que ele, no intuito de simplificar, foi baseado em um modelo com tamanho de janela unitário. Ainda assim o OLI mostrou-se útil.

No teste automático o OLS não apontou falhas.

IV.3 - TESTE DO ALGORITMO MH

A fim de avaliar o uso do testador para o desenvolvimento de um algoritmo que não fosse um protocolo de comunicação foi escolhido um algoritmo de caminho mínimo em grafos (MH - Minimum Hop Distance), proposto por SEGALL [22] (pp

28,29), que é transcrito abaixo. Para este algoritmo foram realizados testes automáticos e manuais.

Constantes e Variáveis do algoritmo para o nó i :

. NUMNODES
Número de nós da rede;

. Gi (l)
Vetor contendo a lista dos l vizinhos do nó i;

. Di (k)
Vetor contendo a distância de i a k para todos os nós k da rede, em número de arcos.

. Pi (k)
Vetor contendo os nós preferenciais para roteamento para alcançar todos os nós k da rede a partir de i;

. Zi
Contador da distancia coberta até o momento pelo algoritmo do nó i;

. Mi
Estado atual do nó i : Work - executando o algoritmo; Normal - algoritmo terminado;

. Ni (l)
Vetor contendo o nível da última mensagem recebida do vizinho l, para todo l pertencente a Gi.

Mensagens enviadas ou recebidas pelo nó i :

.MSG (LISTi)
Mensagem enviada pelo nó i, cujo conteúdo é a lista LISTi;

.MSG (l , LIST)
Mensagem recebida pelo nó i do vizinho l (arco (i,l)), cujo conteúdo é a lista LIST.

.START
Mensagem de disparo do algoritmo, recebida pelo nó i.

Algoritmo MH :

```
PARA TODO k DO
   Pi(k) := NIL;
   Di(k) := NUMNODES;
END;
PARA TODO m PERTENCENTE A Gi DO
   Ni(l) := -1
END;
FOR START OR MSG(l,LIST)DO
   IF Zi = -1 THEN
      Di(i) := 0;
      Mi := Work;
      Zi := 0;
      LISTi := {i};
      ENVIA MSG(LISTi) PARA TODOS OS VIZINHOS DE i;
   END;
   IF MSG AND Mi=Work THEN
      Ni(l) := Ni(l) + 1;
      PARA TODO k PERTENCENTE A LIST DO
         IF Di(k) > Ni(l) + 1 THEN
            Di(k) := Ni(l) + 1;
            Pi(k) := l;
         END;
      END;
      IF PARA TODO m PERTENCENTE A Gi, Zi <= Ni(m) THEN
         Zi := Zi + 1;
         LISTi := { TODO k TAL QUE Di(k) = Zi };
         ENVIA MSG ( LISTi ) PARA TODOS OS VIZINHOS DE i;
         IF ( LISTi É UMA LISTA VAZIA ) THEN
            Mi := Normal;
```

```
          END;
       END;
    END;
END;
```

IV.3.1 - IMPLEMENTAÇÃO

O processo NODE implementa o programa principal do MH. O restante da implementação corresponde às rotinas de biblioteca de sistema, que foram baseadas nas do PROTOCOL5.

O módulo INTERFACE do MH é análogo ao do PROTOCOL5. Sua única diferença reside nos formatos das mensagens.

. As rotinas GETMSG e SNDMSG (análogas a GETF e SENDF do PROTOCOL5) provêm a comunicação através do nível inferior. A recepção de mensagens pelo nível inferior é tambem tratada pelo processo NITASK.
. A recepção de mensagens através do nível superior não é necessária. A transmissão, implememtada por TOHOST, só é utilizada quando a IST atinge o fim do algoritmo. Nesta ocasião a IST envia ao nível superior as listas P e D que ela obteve; com isto, estas ficam registradas na lista de eventos e podem ser examinadas a posteriori pelo usuário para facilitar a depuração.

O tratamento de eventos é feito da mesma forma que no PROTOCOL5. O único evento válido é a recepção de mensagem pelo nível inferior.

A instalação da IST é idêntica à do PROTOCOL5.

A seguir são descritos os formatos das mensagens do MH :

a) NÍVEL INFERIOR

As listas de nós das mensagens do nível inferior são implementadas por vetores :

```
LISTTYPE = RECORD
              NUM : CARDINAL;
              NODES : ARRAY [0..NUMNODES-1]OF CARDINAL;
           END;
```

Onde :

. NUM
Numero de nós da lista.

.NODES
Vetor contendo as identificações de NUM nós. As indentificações dos nós são números no intervalo 0..NUMNODES-1.

As mensagens do nível inferior tem o formato :

```
MSGKIND=(START,DATA);
MSGTYPE = RECORD
             KIND : MSGKIND;
             SRCADD : CARDINAL;
             INFO : LISTTYPE;
          END;
```

Onde :

. KIND
Tipo da mensagem. Os tipos de mensagens de nível inferior válidas estão no "set" MSGKIND e são :

a) START - Disparo do ínicio do algoritmo;

b) DATA - Mensagem contendo lista de nós.

. SRCADD
Endereço da porta inferior do nó fonte da mensagem (este campo só é válido para mensagens do tipo DATA);

. INFO
Lista de nós que constitui a informação útil da mensagem (este campo só é válido para mensagens do tipo DATA).

b) NÍVEL SUPERIOR

As mensagens do nível superior têm o formato :

```
HOSTMSGKIND=(DISTANCE,PREFNEIGHBOR);
HOSTMSGTYPE = RECORD
                KIND : HOSTMSGKIND;
                SRC : CHAR;
                INFO : ARRAY[0..NUMNODES-1]OF CHAR;
          END;
```

Onde :
. KIND
Tipo da lista. Os tipos válidos estão no "set" HOSTMSGKIND e são :

a) DISTANCE - Envio da lista D ao nível superior;

b) PREFNEIGHBOR - Envio da lista P.

. SRCADR
Identificação do nó fonte da mensagem;

. INFO
Vetor que contém a informação útil da mensagem.

A lista interna do INTERFACE, usada para o enfileiramento das mensagens recebidas pelo nível inferior, tem elementos com formato correspondente a LISTTYPE, que é o formato de representação das listas de nós para o programa.

IV.3.2 - TESTE MANUAL

No MH o número de mensagens trocadas entre as ISTs depende do número destas e da topologia da rede, e pode ser muito grande. Além disso, este algoritmo não comporta exceções, pois supõe que existe um nível inferior que provê a transmissão de datagramas de forma confiável.

Assim, em princípio, é pouco prático fazer o teste de forma totalmente manual, editando todas sequências de entradas.

A única entrada manual válida é o disparo do início do algoritmo - mensagem do tipo START. Para iniciar o teste é necessário que o usuário edite esta mensagem e a envie para qualquer das ISTs.

A partir do disparo as ISTs não devem sofrer interferência externa, elas evoluem naturalmente no algoritmo, trocando mensagens entre si, até atingir o seu término. Qualquer interferência do usuário pode introduzir comportamento errôneo.

Portanto o procedimento de teste manual consiste no disparo do algoritmo, seguido pela execução passo a passo realizada paralelamente à inspeção visual do conteúdo das mensagens trocadas pelas ISTs, através de seu "tracing".

Se o número de mensagens fôr muito grande, é conveniente, a fim de facilitar o trabalho do usuário, selecionar o "tracing" de apenas um subconjunto de mensagens.

Este teste foi executado com o auxílio do sistema de depuração MODULA2.

Foram detectados diversos erros de implementação, listados abaixo. Não foram encontrados erros de concepção.

a)
. EFEITO
As mensagens de nível inferior visualizadas nas listas de eventos continham caracteres inválidos.
. CAUSA
Módulo INTEFACE. Erro no preenchimento (rotina SNDMSG) do campo de tamanho das UDs enviadas pelo nível inferior.

b)
. EFEITO
A primeira IST a atingir o fim do algoritmo fazia o sistema bloquear e recair em "range error".
. CAUSA
Processo NODE. Erro nos limites de comandos iterativos para o preeenchimento das listas (P e D) a serem enviadas pelo nível superior ao fim do algoritmo.

c)
. EFEITO
A primeira IST a atingir o fim do algoritmo fazia o sistema bloquear e recair em "range error".
. CAUSA
módulo INTERFACE. Erro no preenchimento (rotina TOHOST) campo de endereço destino nas UDs enviadas pelo nível superior.

d)
. EFEITO
A lista P, passada pelas ISTs ao hospedeiro no fim do algoritmo, apresentava erro no nó preferencial correspondente à IST fonte da mensagem.
. CAUSA
Processo NODE. Erro na inicialização da lista P.

e)
. EFEITO
À recepção de uma lista vazia por qualquer IST o sistema bloqueava e recaía em "cardinal overflow" (erro do RTS - MODULA2).

. CAUSA
Processo NODE. Falta de teste de lista vazia à recepção de mensagens pelo nível inferior.

O teste manual foi realizado para diversas topologias de rede, mostradas na figura IV-3.

IV.3.3 - TESTE AUTOMÁTICO

Existe uma prova formal, apresentada por SEGALL [22] no apêndice (pp 34) , da correção do algoritmo MH. Esta prova é baseada em invariantes lógicos sobre suas variáveis e sobre as interações realizadas.

Partimos do princípio de que a implementação estará correta se os invariantes forem tambem válidos para ela. Assim, o observador foi sintetizado a partir destes invariantes. O conjunto de invariantes foi tomado como uma especificação funcional.

A técnica consiste em utilizar os observadores para testar a validade destes invariantes para uma determinada IST. O acesso às variáveis é feito através do mecanismo de sondas.

As propriedades testadas foram :

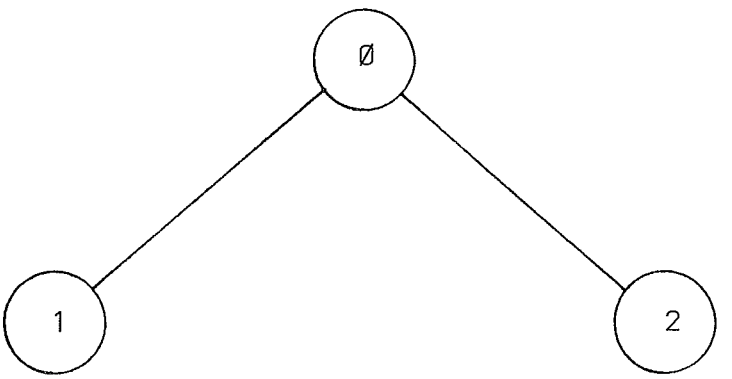
a)
Um nó i envia mensagens se e somente se o seu contador Zi é incrementado simultaneamente (Lema MH1, item b, parcialmente);

b)
Os contadores Zi e Ni somente mudam de valor por incrementos de um (Lema MH1, item c);

c)
Para todo m pertencente ao conjunto de vizinhos de um nó i (Gi (k)), vale Ni(m) = Zi ou Ni(m) = Zi + 1 ou Ni(m)

a)

b)



c)

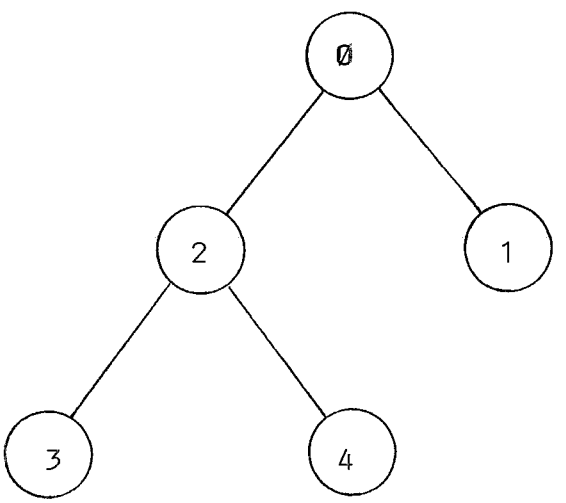
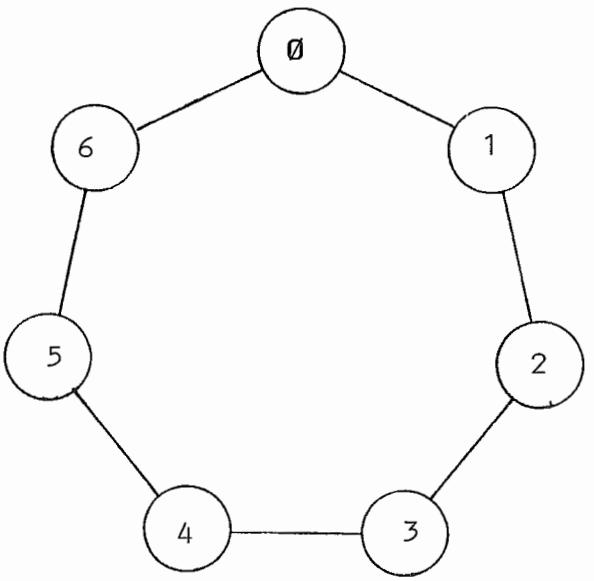


Figura IV-3(a,b,c) - Topologias usadas no Teste Manual do alg. MH

d)



e)

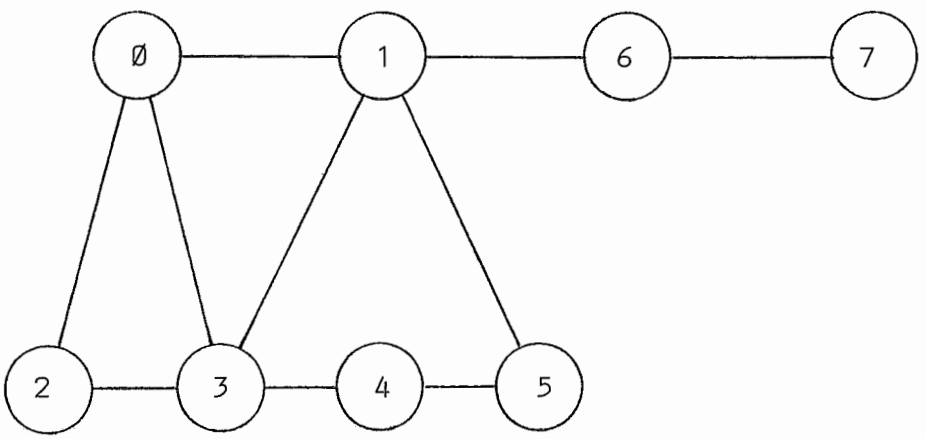


Figura IV-3(d,e) - Topologias usadas no Teste Manual do alg. MH

= Zi - 1 . Alem disto existe ao menos um m tal que Ni(m) = Zi - 1 (Lema MH1, item c);

d)
Seja S a maior distancia-em-arcos ("hop-distance") do nó i na rede; i.e. o nó i possui nós à distância S , mas não a S + 1; Então o nó i irá atingir Zi = S + 1 ao mesmo tempo em que envia listas vazias aos seus vizinhos, terminado o algoritmo. A partir disto Zi não mais é incrementado (Lema MH2, item b).

Os itens e e f do Lema MH1 são testados indiretamente, pois são casos particulares do item d.

Para o teste automático destas propriedades foi utilizado apenas o OLI. o OLS, o CT, o OG e o NI não são necessários. As variáveis sondadas são os contadores Zi e Ni.

As mensagens de erro são geradas pelo OLI nas seguintes condições :

. "ERRO NO ESTADO INICIAL"
Se a IST envia alguma mensagem antes de receber a mensagem de disparo.

. "ERRO NO CONTADOR Z"
Se a IST incrementa o contador Zi sem enviar aos seus vizinhos as mensagens correspondentes (assertiva a).

. "ERRO I NO VETOR N"
Se existe algum m tal que Ni(m) <> Zi e Ni(m) <> Zi + 1 e Ni(m) <> Zi-1 (ass. c).

. "ERRO II NO VETOR N"
Se não existe algum m tal que Ni(m) = Zi - 1 (ass. c).

. "ERRO NA FINALIZAÇÃO"
Se ao término do algoritmo (envio de listas vazias a seus vizinhos) A IST tem Zi <> Si + 1 (ass. d)

. "ERRO NO ESTADO NORMAL"
Se após o término do algoritmo o valor de Zi é incrementado ou alguma mensagem é enviada pela IST (ass. d).

O algorimo do OLI para uma ISTn genérica é apresentado abaixo. Ele é implementado por meio de uma máquina de estados, cujos estados (correspondentes ao estados do MH) são :

. INIC
Estado inicial, antes da ISTn receber o disparo. O OLI é inicializado neste estado.

. WORK
Enquanto a ISTn está executando o MH;

. NORMAL
Após a ISTn terminar o MH.

```
PROCEDURE OLI;
BEGIN;
   INICIALIZA O CONTADOR DE MENSAGENS DA ISTn E O
                                              FLAG DE FIM DE
ALGORITMO;
   WHILE ( EXISTEM UDs EM Q8 ) DO
      RETIRA UD DE Q8;
      IF ( A UD FOI ENVIADA PELA ISTn )THEN
         INCREMENTA CONTADOR DE MENSAGENS DA ISTn;
         IF (A UD CONTEM UMA MENSAGEM COM UMA LISTA VAZIA)
         THEN SETA FLAG DE FIM DE ALGORITMO; END;
      END;
      INSERE UD EM Q6;
      CASE ( ESTADO DA ISTn ) OF
         INICIAL :
            IF (CONTADOR DE MENSAGENS ENVIADAS
                PELA ISTn > 0)
            THEN REGISTRA " ERRO NO ESTADO INICIAL "; END |
         WORK :
```

```
            IF (O NÚMERO DE MENSAGENS ENVIADAS NÃO
                 CORRESPONDE AO INCREMENTO DO CONTADOR Zi)
            THEN REGISTRA " ERRO NO CONTADOR Z "; END;
            IF (EXISTE ALGUM ELEMENTO DO VETOR Ni
                COM VALOR INVÁLIDO)
            THEN REGISTRA " ERRO I NO VETOR Ni "; END;
            IF (NÃO EXISTE ALGUM ELEMENTO NO VETOR Ni
                IGUAL A Zi-1)
            THEN REGISTRA " ERRO II NO VETOR Ni "; END;
            IF ( O FLAG DE FIM DE ALGORITMO ESTÁ SETADO )
            THEN
               IF ( O VALOR DE Zi NÃO É VÁLIDO )
               THEN REGISTRA " ERRO NA FINALIZAÇÃO "; END;
               ESTADO DA ISTn PASSA PARA NORMAL;
            END|
         NORMAL :
            IF ( O VALOR DE Zi FOI MODIFICADO )
               OR
               ( ALGUMA MENSAGEM FOI ENVIADA PELA ISTn )
            THEN REGISTRA " ERRO NA FINALIZAÇÃO " END;
      END; (* CASE *)
   END; (* WHILE *)
   WHILE ( EXISTEM UDs EM Q7 ) DO
      RETIRA UD DE Q7;
      IF ( A UD É DESTINADA À ISTn )
         AND
         ( ESTADO DA ISTn É INIC )
      THEN ESTADO DA ISTn PASSA A WORK; END;
      INSERE UD EM Q9;
   END;
END OLI;
```

Foram realizados testes automáticos para duas topologias :

. Figura IV-3c, na IST correspondente ao nó 2;

. Figura IV-3d, na IST correspondente ao nó 3.

Neles o OLI não detectou nenhuma falha.

CAPÍTULO V

CONCLUSÃO

CAPÍTULO V

Neste capítulo são analisados os resultados dos testes experimentais realizados com o auxílio do testador. A partir disto o testador é avaliado e são propostas extensões a fim de incrementar a sua capacidade.

V.1 - RESULTADOS DOS TESTES - AVALIAÇÃO

V.1.1 - IMPLEMENTAÇÃO E INSTALAÇÃO

As ISTs utilizadas no testador podem, em princípio, ser implementadas em qualquer sistema de desenvolvimento que gere código para a família de microprocessadores INTEL8088/86, em formato executável DOS.

As implementações usadas como ISTs diferem de implementações reais em três aspectos (desde que estas sejam implementadas no mesmo ambiente daquelas) :

. Módulo INTERFACE;

. Procedimento de instalação e inicialização;

. Restrições de E/S, de interrupções e de memória dinâmica.

A confecção do módulo INTERFACE não apresenta maiores dificuldades, desde que o ambiente em que são desenvolvidas as ISTs permita a realização dos processamentos de baixo nível necessários :

. A transmisão pode ser implementada por meio de acesso direto aos serviços do testador.

. A recepção pode ser implementada por meio de processos que executam a cada ciclo de relógio e esvaziam as filas de entrada do testador, tratando as UDs lá contidas de forma conveniente.

O tratamento das outras restrições não é trivial e requer do usuário maior controle sobre o ambiente em que ele desenvolve suas implementações.

Nossa equipe utiliza dois ambientes para o desenvolvimento de "software" para sistemas distribuídos :

. Sistema MODULA2, já descrito;

. Sistema INTEL (PLM86, ASM86, LINK86, RUN-ACCESS, etc), para o qual temos também um pacote de "software" básico modular e voltado para aplicações em tempo real, análogo ao pacote de "software" básico MODULA2.

Uma vez que temos controle sobre o "software" básico utilizado, não temos dificuldade em adaptar nossos programas para funcionarem como ISTs e vice-versa. Na verdade, nos ambientes de desenvolvimento que utilizamos, as modificações necessárias nos programas são de pequena monta.

A adpatação de implementações desenvolvidas em outros sistemas não foi investigada.

Nos testes demonstrativos foi adotada a fatia-de-tempo de 54msec, que é o período de relógio do IBM-PC.

As rotinas de interrupção das ISTs são ativadas a cada fatia, assim o seu período também é de 54msec. Este é um padrão PC, de forma que, em princípio, todo o "software" desta máquina, inclusive o "software" básico utlizado, assume este período de interrupção de relógio.

A escolha da fatia tem as seguintes justificativas :

. Por simplicidade adotamos o período padrão do relógio do IBM-PC.

. As ISTs empregadas nos testes de algoritmos que visavam a

demonstração e avaliação do testador não estavam associadas a sistemas-alvo reais. Portanto não existiam requisistos de tempos de resposta, etc, que conduziriam ao dimensionamento das fatias. De qualquer forma tais requisitos naõ seriam sisgnificativos, visto que os algoritmos testados são voltados para redes de baixa velocidade, ou seja, assumem que o serviço de comunicação (nível inferior) é lento. Assim podemos assumir que o serviço é lento o suficiente para que a fatia não introduza distorções.

. Nos testes lógicos preliminares que iniciamos em outros algoritmos (ver item V.3), adaptamos as ISTs, redimensionando time-outs, etc, para garantir que o uso do testador não introduza erros. Para tais testes é (pelo menos preliminarmente) aceitável descartar parâmetros relacionados ao desempenho, que conduziriam ao dimensionamento da fatia.

. Não previmos a realização de testes de desempenho no testador proposto. Em tais testes seria imperativo o correto dimensionamento da fatia.

Para os casos em que seja necessário o dimensionamento das fatias, bem como dos períodos de ativação das rotinas de relógio da ISTs, é desejável que esta configuração possa ser realizada confortavelmente através da IHM.

V.1.2 - TESTES MANUAIS

Nos testes manuais detectou-se, com facilidade, uma grande quantidade de erros de programação, tais como : erros na manipulação de apontadores, erros em limites de comandos iterativos, etc.

Para a detecção destes erros simples, as ISTs (que implementavam algoritmos também simples) foram exercitadas por sequências geradas empiricamente pelo usuário,

compostas por pequenos números de UDs.

A detecção de erros mais complexos requer, em princípio, o uso de sequências mais longas, cuja obtenção não é intuitiva.

O mesmo ocorre com relação ao teste de algoritmos mais elaborados.

Nestes casos recai-se no problema de geração de sequências de teste.

Na prática, o poder de teste manual é limitado pelo número de entradas (saídas) que devem ser geradas (avaliadas) manualmente pelo usuário. Se este número for grande o teste manual pode ser impraticável.

A experiência indica que os testes manuais devem ser usados para a detecção de falhas de conformidade relacionados a erros de programação e erros de lógica simples (em suma, para depuração inicial de implementações de algoritmos). Erros mais elaborados devem ser detectados através de testes automáticos.

A depuração das ISTs, desenvolvidas no sistema MODULA2, foi agilizada através do uso do RTD em paralelo com o testador. O RTD é uma poderosa ferramenta de depuração, que possui facilidades como : estabelecimento interativo de "breakpoints", consulta e alteração de variáveis de programa, etc.

Não foram testadas ISTs desenvolvidas em outras linguagens. Nestes casos, uma vez que não é possível utilizar o RTD, deve ser considerada a possiblibidade de usar outros sistemas de depuração (DEBUG, etc.) junto com o testador.

A grande dificuldade encontrada no uso da IHM é relacionada à representação hexadecimal das áreas de dados das UDs. O emprego desta representação para a edição de UDs e para a

sua apresentação na lista de eventos não é confortável e exige atenção redobrada do usuário, pois pode conduzir a erros.

V.1.3 - TESTES AUTOMÁTICOS

O teste automático é a segunda etapa do teste de uma implementação. Ele deve ser precedido por uma etapa de teste manual, onde é realizado um teste básico de conformidade.

A síntese de observadores é a base para a realização de testes automáticos.

O princípio de funcionamento dos observadores consiste em testar um modelo reduzido do algoritmo, que prevê um determinado conjunto de erros. O objetivo do teste automático é simplesmente responder à seguinte pergunta : Algum dos erros previstos ocorre ?

A determinação das sequências de interações que conduzem aos erros deve ser feita manualmente pelo usuário, através da análise das listas de eventos.

Os observadores podem ser construídos empíricamente pelo usuário a partir de uma lista de erros que este deseja verificar. Todavia a sua síntese a partir de um modelo formal é mais indicada, pois assim é possível selecionar de forma controlada as particularidades do modelo que serão testadas.

Os observadores utilizados nas experiências foram obtidos a partir de descrições formais dos algoritmos : modelos em Redes de Petri e invariantes lógicos.

Um exemplo prático de teste automático aleatório, que esclarece acerca da obtenção e do funcionamento dos observadores, é o do PROTOCOL5 . Neste teste o OLI detectou

um erro de projeto que seria difícil de rastrear manualmente, entretanto tambem apontou um erro inexistente.

Este tipo de falha no comportamento dos observadores é previsível. Isto deve-se ao fato de que os dois mecanismos em que se baseiam os observadores para a deteção de erros (o acesso às UDs e às sondas) apresentam limitações, discutidas abaixo, inerentes à arquitetura do testador. Tais limitações devem devem ser levadas em conta na interpretação dos resultados obtidos.

. UDs

A observação de UDs tem duas incertezas implícitas : não é possível prever o momento em que as UDs serão realmente recebidas e tratadas pelas ISTs; não é possível prever a ordem relativa de recepção e tratamento das UDs de nível superior e inferior.

Estas incertezas são devidas a dois fatores : a estratégia, empregada no testador, de distribuição das UDs para as ISTs através de filas de entrada, aliada à falta de um mecanismo pelo qual o testador sinalize as ISTs da chegada de UDs; e a imprevisibilidade dos mecanismos internos de recepção e tratamento das ISTs.

. Sondas

A incerteza introduzida pelo mecanismo de sondas está relacionda à incerteza das UDs, pela qual não é possível prever o momento nem a ordem relativa em que o tratamento das UDs recebidas irá se refletir nos valores das variáveis das ISTs. Isto é agravado pelo fato de que as ISTs são interrompidas pelo testador em instantes arbitrários.

É difícil, portanto, correlacionar os valores das sondas com as UDs que circulam através das interfaces das ISTs.

Supreendentemente nas experiências com o MH não enfrentamos problemas desta ordem, tavez devido à simplicidade do algoritmo.

O mecanismo de sondas parece bastante restrito : as propriedades testadas devem, em princípio, ser bastante gerais.

O observador global não foi avaliado, mas o bom-senso indica que é análogo aos observadores locais. As propriedades que ele deve testar são relaciondas às UDs dos dois níveis, p. ex. no PROTOCOL5 toda recepção de mensagem (nível superior) causa o envio de um quadro (nível inferior) e vice-versa.

O uso do NI não foi avaliado.

V.2 - EXTENSÕES

Nos próximos itens são sugeridas extensões imediatas do trabalho do testador, com vistas a incrementar a técnica de teste. Primeiramente são discutidos os testes manuais e a seguir os automáticos.

Uma primeira sugestão, que visa reduzir o tempo gasto nos testes, é a utilização de uma arquitetura distribuída, preferencialmente multiprocessador (ou ainda uma rede local).

Neste caso a máquina em que executa o testador pode ser qualquer, entretanto as máquinas em que executam as ISTs devem ser idênticas às máquinas do sistema-alvo.

A versão de testador para tal sistema, infelizmente fora do nosso alcance, é conceitualmente idêntica à proposta. Ela, entretanto, atribuiria simultâneamente fatias-de-tempo às ISTs nos diversos processadores.

Uma vez que a execução das ISTs seria simultânea, e não sequencial, ganharíamos em desempenho, abreviando os tempos gastos nos testes.

V.2.1 - TESTES MANUAIS

São aqui descritas informalmente, de forma resumida, extensões que visam eliminar dificuldades encontradas no uso do testador e incrementar sua capacidade para testes manuais.

. REPRESENTAÇÃO ESTRUTURADA DE UDs

É desejável substituir o formato de representação hexadecimal dos campos de dados das UDs na IHM por uma representação estruturada. Para isto deve ser criado um módulo específico.

Este módulo deve conhecer os formatos, préviamente definidos pelo usuário, de todas as mensagens possíveis no sistema, e fornecer aos demais módulos do testador primitivas para a edição e a exteriorização destas estruturas.

O conjunto de formatos de mensagens pode ser definido com o auxílio de uma linguagem (e respectivo compilador) voltada para a declaração de tais estruturas (ex : ASN.1 - " Abstract Syntax Notation One"), e pode ficar diponível para o testador em arquivos de configuração.

A primitiva de apresentação deve ser capaz de interpretar o conteúdo de uma mensagem e apresentá-lo em vídeo num formato confortável para o usuário : o tipo da mensagem deve ser identificado; os campos devem ser rotulados por seus nomes; os campos simbólicos devem ter seus conteúdos traduzidos; etc.

A facilidade de edição deve permitir ao usuário a entrada de UDs, através da IHM, de forma equivalente.

A sugestão é implementar um codificador/decodificador ASN.1, a exemplo de MURALIDHAR [28].

. ACOMPANHAMENTO

Uma das vantagens do uso do RTD na depuração é permitir o acesso do usuário às variáveis das ISTs.

O acompanhamento visa fornecer parcialmente ao usuário, através da IHM, esta facilidade, o que é essencial para a depuração de ISTs desenvolvidas em outros ambientes.

O acompanhamento é feito através de um conjunto de telas (janelas) de vídeo, selecionáveis a partir da IHM, onde, além de uma parte estática informativa, são apresentados em tempo real (atualizados a cada varredura das ISTs) os conteúdos de sondas das ISTs.

O conjunto de telas deve ser totalmente configurável, p. ex. através de arquivos, e a sua definição pode ser auxiliada por editores "off-line" que gerem os arquivos correspondentes.

Uma opção a mais é permitir ao usuário alterar manualmente o conteúdo das variáveis sondadas, mas isto parece ser de pouca ou nenhuma utilidade.

. REGISTRADOR DE EVENTOS

O RE disponível atualmente provê uma facilidade de "tracing" bastante limitada. Ele registra os eventos de transmissão e recepção de UDs segundo as opções configuradas nas palavras de status das ISTs. Estas são

modificáveis pelo usuário a partir da IHM.

É possivel definir uma série de extensões para o RE, cuja conveniência é proporcional à complexidade dos algoritmos em teste. Estas extensões, que dão ao RE opções análogas às disponíveis em analisadores de estados lógicos e analisadores comerciais de protocolos, facilitam o trabalho de análise das listas de eventos, pois evitam que estas contenham grandes quantidades de dados inúteis.

Uma opção interessante consiste em selecionar, a partir de um conjunto de restrições, os eventos a serem registrados. Alguns exemplos de restrições possíveis são :

. Restrições de campo e valor
Somente são registrados os eventos correspondentes às UDs cujo conteúdo de determinados campos estiverem em intervalos prefixados.

. Restrições de número
É registrado um número predefinido de eventos.

. Restrições de tempo
São registrados apenas os eventos que ocorrerem em uma determinada janela de tempo virtual.

. Restrições de endereçamento
São registrados apenas os eventos correspondentes a mensagens que forem destinadas a determinadas ISTs e/ou geradas por determinadas ISTs.

Combinações de restrições, do mesmo tipo e/ou de tipos diferentes, podem ser selecionadas interativamente pelo usuario a partir da IHM , ou ainda configuradas a tempo de inicializacao por meio de arquivos.

O caso limite de sofisticação deste mecanismo é a definição de uma linguagem para a descrição, através de sentenças lógicas que encerrem tais restições, dos eventos que devem

ser registrados. Esta linguagem seria interpretada pelo testador.

Outra opção que pode ser conveniente é a adoção de um mecanismo de disparo ("trigger") do registro de eventos. O registro só seria iniciado, instantaneamente ou com retardo ("delayed"), a partir da detecção de um evento em particular (ou um evento entre um conjunto de opções possíveis) configurado(s) através de restrições.

Para a implementação tais opções sofisticadas para o RE, é desejável a disponibilidade das facilidades para o tratamento de mensagens de forma estruturada.

. APLICAÇÃO DE SEQUENCIAS DE TESTE

Para poupar ao usuário o trabalho de edição "on-line" de longas seqências de UDs, é desejavel a disponibilidade recursos de IHM que permitam comandar a aplicação de sequências de UDs previamente geradas, contidas em arquivos.

A geração dos arquivos de sequências de UDs pode ser realizada por meio de editores estruturados "off-line".

Para esta abordagem é importante o tratamento das UDs de forma estruturada.

A aplicação de uma sequência pode ser feita passo a passo ou de forma automática. No primeiro caso as UDs são lidas do arquivo e despachadas uma a uma, sob comando do usuário; no segundo as UDs devem estar rotuladas com os tempos virtuais de sua aplicação, que é feita automaticamente pelo testador.
O aplicador de sequências de teste é um utilitário genérico. Ele não é útil somente para testes manuais, mas também para testes automáticos por sequência de teste.

V.2.2 - TESTES AUTOMÁTICOS

A capacidade de deteção de erros em testes automáticos esbarra em restrições inerentes à arquitetura do testador, que não podem, em princípio, ser eliminadas.

Assim, numa primeira instância, o aprimoramento dos testes automáticos não implica em desenvolvimentos no testador e sim na investigação de técnicas para, a partir de modelos formais de algoritmos :

. Sintetizar observadores;

. Obter sequências de teste (isto é tambem importante para o aprimoramento dos testes manuais).

V.3 - NOVOS TESTES

Atualmente estão sendo realizados trabalhos de adaptação e instalação de dois algoritmos reais, para a execução de testes :

. O protótipo do algoritmo de difusão confiável [], desenvolvido em MODULA2, que já foi testado manualmente em uma versão preliminar do testador, está sendo adaptado para esta nova versao. O objetivo é a realização de novos testes manuais e de testes automáticos.

. A rede local CEPEL (já descrita na introdução - CAPÍTULO I), implementada no sistema de desenvolvimento INTEL, e o "software" básico correspondente, estão sendo adaptados para testes. A arquitetura definida para os testes, mostrada na figura V-1, é constituída por cinco nós

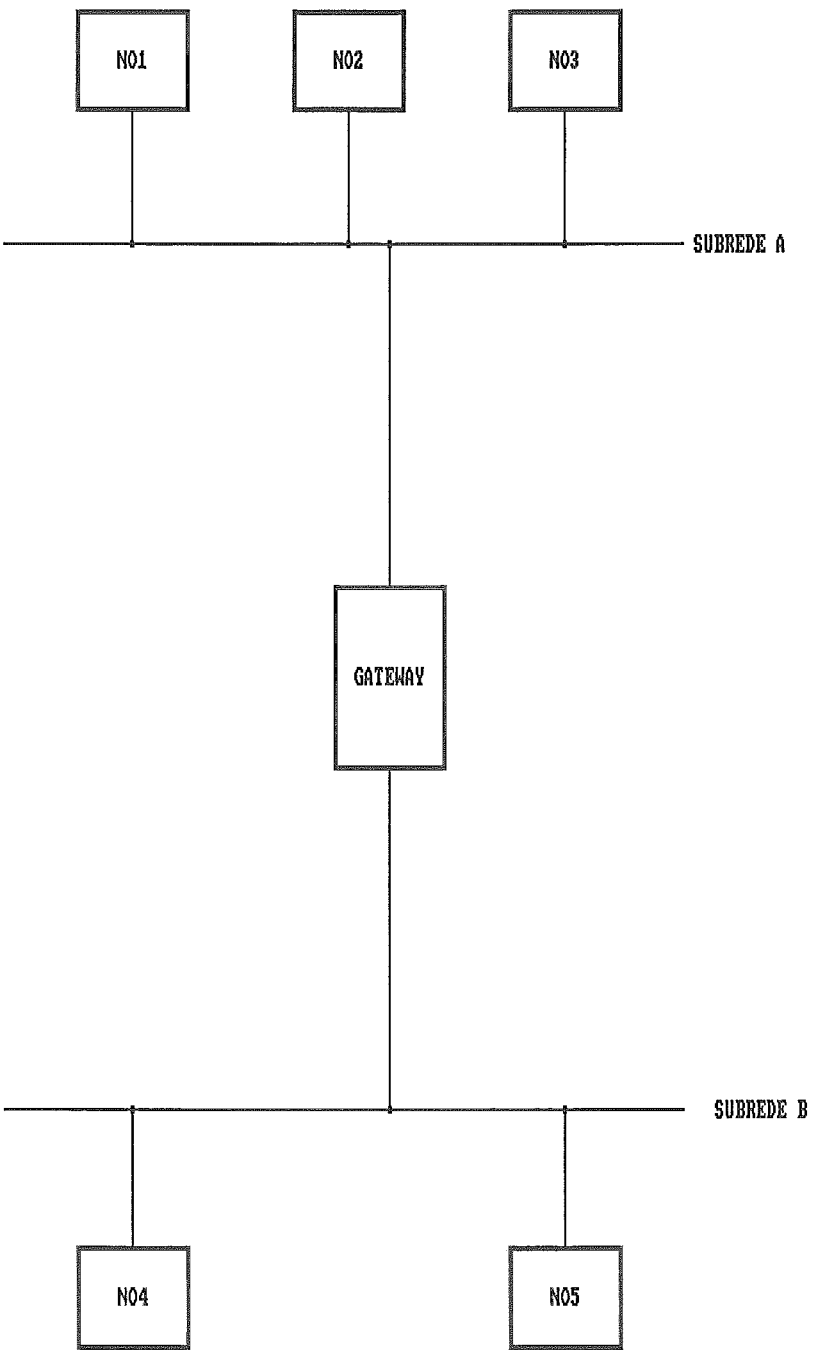


Figura V-1 - Teste da Rede Local CEPEL

dispostos em duas subredes locais que são interligadas por um "gateway". Este trabalho tem dois objetivos : testar o programa aplicativo do "gateway"; e tornar o testador disponível para o teste de ISTs desenvolvidas no sistema INTEL, o que aumenta o seu universo de aplicações.

V.4 - CONCLUSÃO

A necessidade de projetar e implementar um algoritmo distribuído complexo, que é o protocolo de difusão confiável, colocou em evidência as técnicas de projeto de tais algoritmos. Particularmente importantes mostraram-se as técnicas de teste de implementações.

A estratégia tradicional que empregávamos, que consiste em testar as implementações exaustivamente, de forma manual, no sistema-alvo, revelou-se inadequada, pois era necessário reproduzir controladamente um grande número de situações possíveis.

Para a obtenção de um produto confiável em menor tempo, tornou-se imperativa a construção de uma ferramenta que auxiliasse o teste. Optou-se por construir um testador que substituisse o sistema-alvo e fornecesse um ambiente controlado de teste.

O testador que foi construído permite a realização de testes manuais e automáticos em conjuntos de implementações. Este testador foi depurado e avaliado através de testes de algoritmos simples.

A partir das experiências realizadas foi proposto um conjunto de extensões que visam aumentar a sua capacidade de teste e eliminar dificuldades encontradas no seu uso.

Apesar de apresentar restrições em relação às ISTs e limitações em sua capacidade de teste, o testador mostrou-se uma ferramenta adequada para o teste de algoritmos

implementados por nossa equipe, pois é compatível com os ambientes de desenvolvimento que utilizamos.

Atualmente estamos iniciando o uso do testador para o desenvolvimento de algoritmos reais.

BIBLIOGRAFIA

[1] BOCHMANN, G. v.; Usage of Protocol Development Tools: The Results of a Survey; Seventh IFIP International Meeting on Protocol Specification, Testing and Verification; Zurich; may 5-8 1987; I2.

[2] COST 11 BIS GROUP on FDT/AEFSD (Formal Description Techniques / Architecture and Extended Finite State Descriptions); Fifth IFIP WG.6.1 International Workshop On Protocol Specification, Verification and Testing; Toulouse; june 10-13 1985; pp 9-21..9-42.

[3] COURTIAT, J. P.; PEDROSA, A. e AYACHE, J. M.; A Simulation Environment for protocol Specifications described in ESTELLE; Fifth IFIP WG.6.1 International Workshop On Protocol Specification, Verification and Testing; Toulouse; june 10-13 1985; pp 6-3..6-22.

[4] FISCHER, W.; SAUER, K. P. e DENZEL, W.; A Simulation Technique for Communication Protocols Based on a Formal Specification by SDL;Fifth IFIP WG.6.1 International Workshop On Protocol Specification, Verification and Testing; Toulouse; june 10-13 1985; pp 7-21..7-35.

[5] GROZ, R.; Correction et Preuve Partielle du Protocole D'Exclusion Mutuelle de Galaxie; Note Technique NT/LAA/SLC/123 CNET LANNION Département EVP; 1983.

[6] GROZ, R. ; JARD, C. e LAUSUDRIE, C.; Attacking a Complex Distributed Algorithm from Different Sides: An Experience with Complementary Validation Tools; Computer Networks and ISDN Systems; North-Holland; vol 10; 1985; pp 245..257.

[7] JARD, C.; Specification et Validation D'Un Algorithime Distribue D'Exclusion Mutuelle - Mise en Oeuvre de la Simulation : Methode et Resultats; Note Technique NT/LAA/SLC/93 CNET LANNION Département EVP; Juillet 1982.

[8] JARD, C.; MONIN, J. e GROZ, R.; Experience in Implementing X.250 (a CCITT subset of ESTELLE) in VEDA - Fifth IFIP WG.6.1 International Workshop On Protocol Specification, Verification and Testing; Toulouse; june 10-13 1985; pp 7-1..7-20.

[9] PETERSON, J. L.; Petri Net Theory and the Modeling of Systems ; Prentice-Hall; 1981; ISBN 0-13-661983-5.

[10] SARIKAYA, B.; Recent Developments in Protocol Testing; Quinto Simpósio Brasileiro de Redes de Computadores; São Paulo; abril 13-15 1987; pp 372..392.

[11] The SPECS Consortium e BRUIJNING, J.; Evaluation and Integration of Specification Languages; Computer Networks and ISDN Systems; North-Holland; vol 13; 1987; pp 75..89;

[12] SILVA, A. J. R.; Gerenciador de Dados Replicados para Centros de Supervisão e Controle Baseados em uma Arquitetura Distribuída ; Tese M.Sc.; COPPE/UFRJ (Coordenação de Projetos de Pós-Graduação em Engenharia / Universidade Federal do Rio de Janeiro); abril 1988.

[13] SOUZA, W. L. e FERNEDA, E.; Aplicações do Compilador ESTELLE; Quinto Simpósio Brasileiro de Redes de Computadores; São Paulo; abril 13-15 1987; pp 107..120.

[14] VENKATRAMAN, R. C. e PIATKOWSKI, T. F.; A Formal

Comparison of Formal Protocol Specification Techniques; Fifth IFIP WG.6.1 International Workshop On Protocol Specification, Verification and Testing; Toulouse; june 10-13 1985; pp 9-1..9-20.

[15] VISSERS, C. A.; Trends and Proliferation of Formal Description Techniques for OSI Standards; Quinto Simpósio Brasileiro de Redes de Computadores; São Paulo; abril 13-15 1987; apêndice.

[16] LOGITECH MODULA2 - Version 3.0 - User's Manual; LOGITECH Inc.; Document Number LU-UD-009-1; Third Edition september 1987.

[17] LOGITECH MODULA2 - Version 3.0 - TOOLKIT; LOGITECH Inc.; Document Number LU-UD-0010-1; August 1987.

[18] WIRTH, N.; Programming in MODULA2; Third, corrected edition; Springer-Verlag - Texts and Monographs in Computer Science; 1985; ISBN 0-387-15078-1 (U.S.).

[19] SALE, A. H. J.; MODULA2 - Discipline & Design; Addison-Wesley Publishing Company; 1986; ISBN 0-201-12921-3.

[20] Arquitetura do STD - Testador do Subsistema de transmissão de dados do SINSC; Eletrobras DOS-DECT; Outubro/1986.

[21] TANEMBAUM, A. S.; Computer Networks; Prentice-Hall Inc.; ISBN 0-13-165183-8.

[22] SEGALL, A.; Distributed Network Protocols; IEEE Transactions on Information Theory; vol. IT-29, No. 1, pp. 23..34; January 1983.

[23] DSSOULI, R. e BOCHMANN, G.; Conformance Testing with Multiple Observers; ; Département d'informatique

et de recherche opérationnelle (I.R.O.); Université de Montréal; C.P 6128; Succ. "A" Montréal; P.Q. H3C 3J7.

[24] BRAMS, G. W.; Réseaux de Petri : Théorie et Pratique - Tome 2 - modélisation et applications; Masson S.A.; 1983; ISBN 2-903-60713-3.

[25] BUDKOWSKI, S. e DEMBINSKI, P.; An Introduction to Estelle : A Specification Language for Distributed Systems; Computer Networks and ISDN Systems; North-Holland; vol. 14, No. 1, pp 3..23; 1987.

[26] BOLOGNESI, T. e BRINSKMA, E.; Introduction to the ISO Specification Language LOTOS; Computer Networks and ISDN systems; North-Holland; vol. 14, No. 1, pp 25..29; 1987.

[27] MOURA, J. A. B.; SAUVÉ J. P.; GIOZZA, W. F. e ARAÚJO, J. F. M.; Redes Locais de Computadores - Protocolos de Alto Nível e Avaliação de Desempenho; Editora McGraw-Hill; 1986.

[28] MURALIDHAR, K. H.; MAP 2.1 Network Management and Directory Services Test System; Seventh IFIP International Meeting on Protocol Specification, Testing and Verification; Zurich; may 5-8 1987; C1.2.

[29] URAL, H.; A Test Derivation Method for Protocol Conformance Testing; Seventh IFIP International Meeting on Protocol Specification, Testing and Verification; Zurich; may 5-8 1987; C1.1.

[30] BUDKOWSKI, S. e DEMBINSKI, P.; Simulating ESTELLE specification with time parameters; Seventh IFIP International Meeting on Protocol Specification, Testing and Verification; Zurich; may 5-8 1987; B3.2.

[31] BOCHMANN, G. v.; DSSOULI R.; SOUZA, W. L.; SARIKAYA, B. e URAL, H.; Use of Prolog for Building Protocol Design Tools; Fifth IFIP WG.6.1 International Workshop On Protocol Specification, Verification and Testing; Toulouse; june 10-13 1985; pp 3-27..3-39.

[32] SARACCO, R. e TILANUS, P. A. J.; CCITT SDL : Overview of the Language and its Applications; Computer Networks and ISDN Systems; North-Holland; vol. 13, pp 65..74, 1987.

APÊNDICE A

MODELO DO PROTOCOL5 EM REDES DE PETRI

APÊNDICE A

Neste apêndice é apresentado o modelo em Redes de Petri do PROTOCOL5 com janela unitária. Este é um dos algoritmos usados na depuração, demonstração e avaliação do testador. Ele é descrito no item IV.2.

A.1 - MODELO DO PROTOCOL5

As convenções adotadas, tanto no modelo deste apêndice quanto no da figura IV-1, são listadas abaixo. A seguir são apresentadas as figuras correspondentes ao modelo.

. CONVENÇÕES DA FIGURA IV-1

a) A entrega de dados para o hospedeiro é modelada pelo lugar ("place") HOSTIN. O recebimento de dados provenientes deste é feita pelo lugar HOSTOUT.

b) Alguns lugares correspondem a condições específicas de contadores e variáveis do algoritmo (ex: NextToSend = 1; NBuffered = 0; etc.), eles são rotulados de forma a explicitar estas condições. Tais condições são verificadas quando da existência de fichas nestes lugares. O glossário dos nomes de variáveis do algortimo é :

i) NTS - NextToSend
ii) AE - AckExpected
iii) FE - FrameExpected
iv) NB - NBuffered

c) O canal é modelado por um conjunto de lugares, correspondentes aos valores que podem ser assumidos pelos diversos campos de controle da mensagem (KIND, SEQ e ACK), que são auto-explicativos :

i) KIND=DATA;
ii) KIND=ACK;
iii) SEQ=0;
iv) SEQ=1;
v) ACK=0;
vi) ACK=1;

A transmissão é feita pelo preenchimento, com fichas, dos "campos" correspondentes ao quadro desejado. Não é permitida a permanência de mensagens pendentes no canal.

d) o modelo completo do sistema, que assume um canal "full-duplex", implica em duplicar o modelo da figura IV-1(a).

. CONVENÇÕES DO APÊNDICE A

a) Convenção (a) da figura IV-1.

b) Convenção (b) da figura IV-1.

c) Rotinas e outros blocos do modelo são definidos em separado e identificados por nomes. Nos locais onde são referenciados, são modelados como lugares rotulados convenientemente.
As rotinas têm sempre um ou mais lugares de entrada e apenas um lugar de saída. Os diversos lugares de entrada correspondem aos parâmetros de chamada possíveis. Eles são rotulados de forma a explicitar isto, usando o nome da rotina e os parâmetros correspondentes.
O lugar de saída é rotulado com o nome da rotina precedido da palavra "RETURN".
Conceitualmente a "chamada" de uma rotina consiste em

colocar uma ficha no lugar de entrada correspondente e aguardar o aparecimento de uma ficha no lugar de saída.
Os blocos são análogos, mas têm apenas um lugar de entrada e um de saída. Eles visam apenas simplificar as figuras dos modelos.

d) O canal é modelado como um conjunto de lugares, um para cada tipo de quadro possível:

i) FRAME 0/0 - Quadro de dados com número de sequência igual a zero (Seq=0), reconhecendo em "piggyback" o quadro de número de sequência zero (Ack=0);
ii) FRAME 0/1 - Quadro Seq=1 Ack=0;
iii) FRAME 1/0 - Quadro Seq=1 Ack=0;
iv) FRAME 1/1 - Quadro Seq=1 Ack=1;
v) ACK 0 - Quadro de reconhecimento em separado do quadro de número de sequência zero (Ack=0);
vi) ACK 1 - Reconhecimento Ack=1.

A transmissão é realizada colocando uma ficha no lugar correspondente ao tipo do quadro desejado.

e) Os temporizadores (ACKTIMER, TIMER0, TIMER1) são modelados por lugares. O temporizador está armado quando existe uma ficha no lugar correspondente.

f) Transições especiais, dependentes do tempo, relacionadas a chamadas temporizadas a WAITMSG, a expiração de temporizadores, etc, são rotuladas convenientemente.

g) A habilitação/desabilitação do hospedeiro é modelada pelo lugar HSENABLE. O hospedeiro está

habilitado quando existe uma ficha neste lugar.

h) A ocorrência de eventos é modelada por meio de lugares rotulados de forma a explicitá-los (EVFRAMEARRIVAL, EVHOSTIDLE, EVTIMEOUT, EVHOSTREADY). A sinalização de um evento consiste em colocar uma ficha no lugar correspondente.

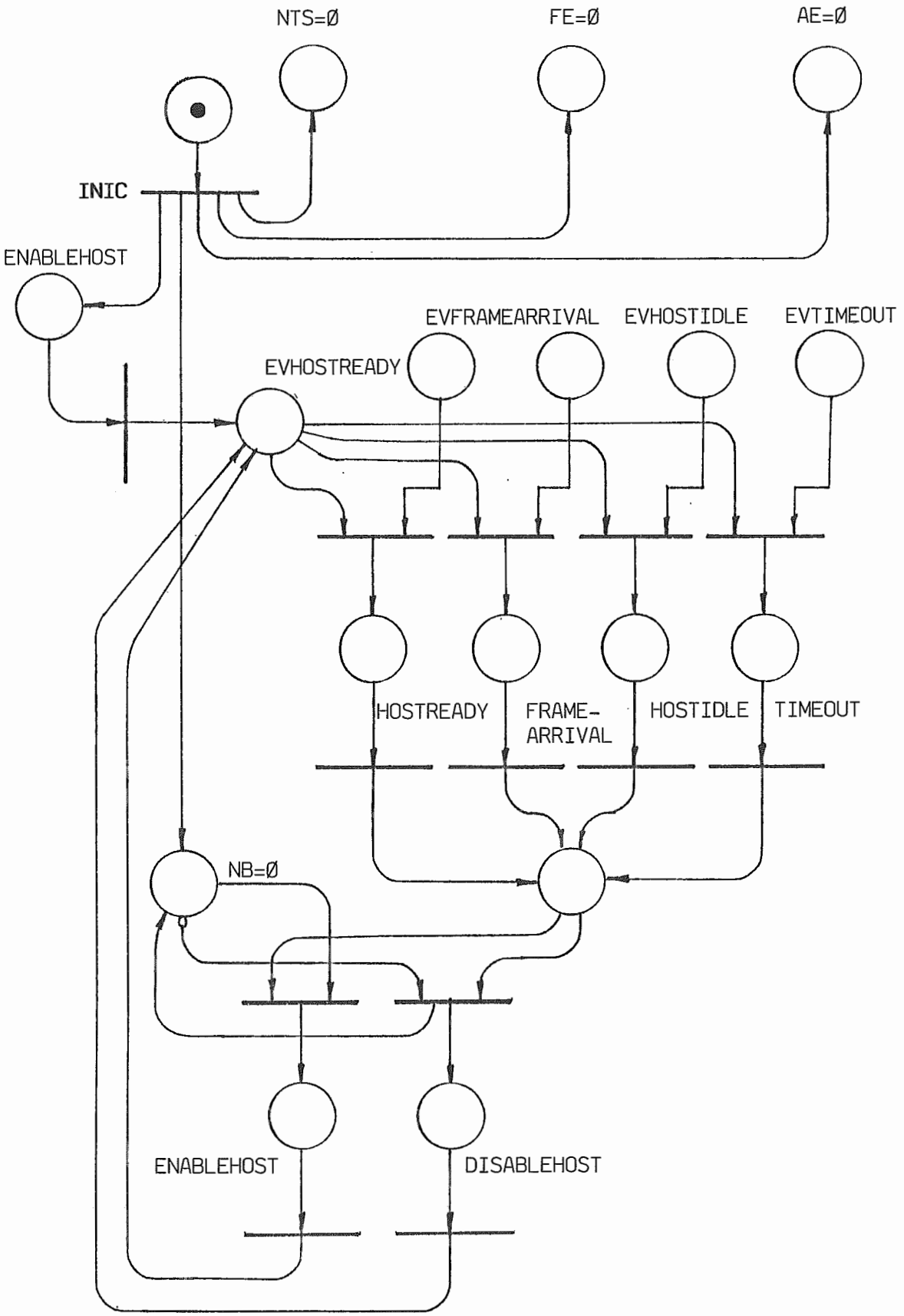


Figura A-1 - Processo PROTOCOL5

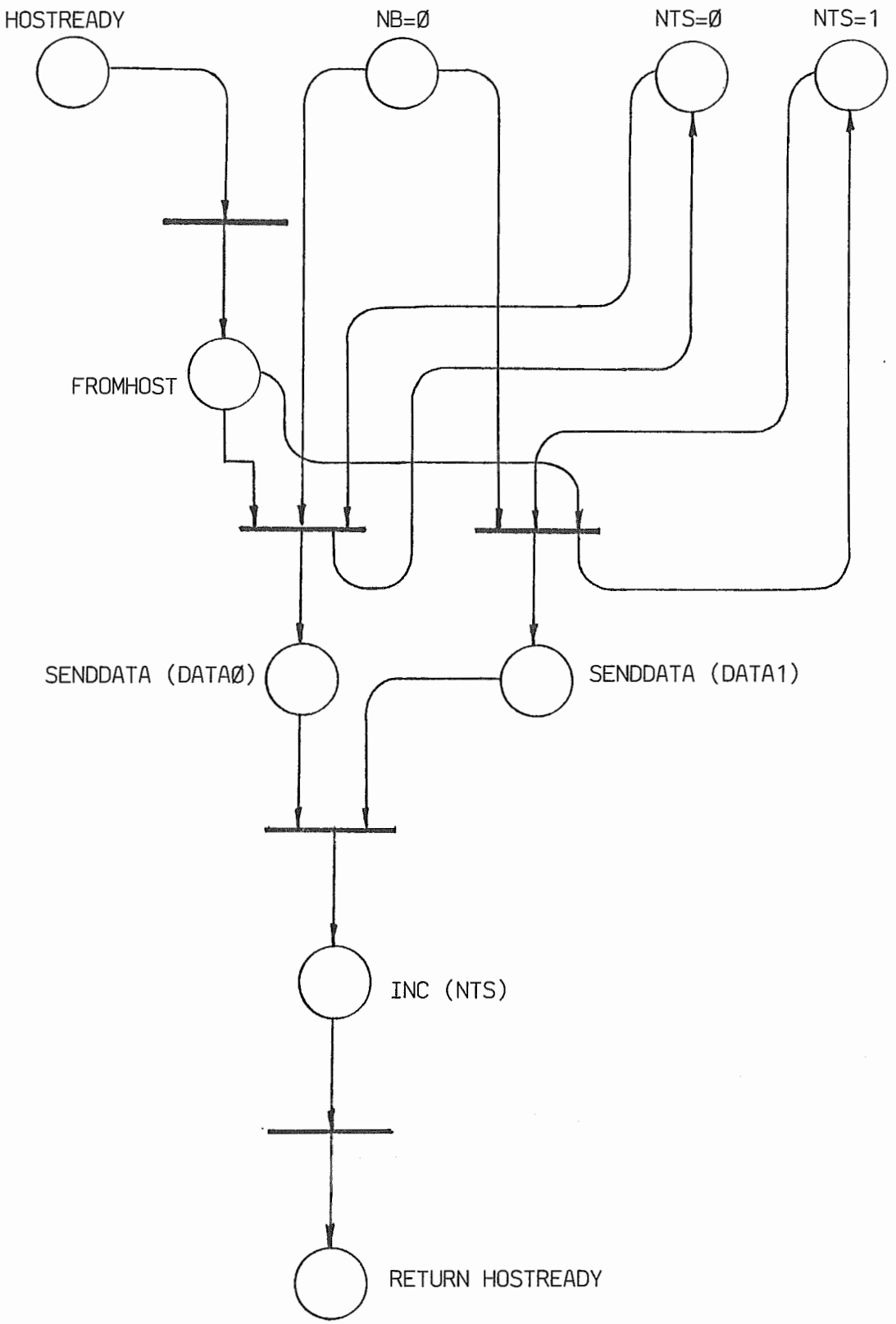


Figura A-2 - Tratamento do Evento HOSTREADY

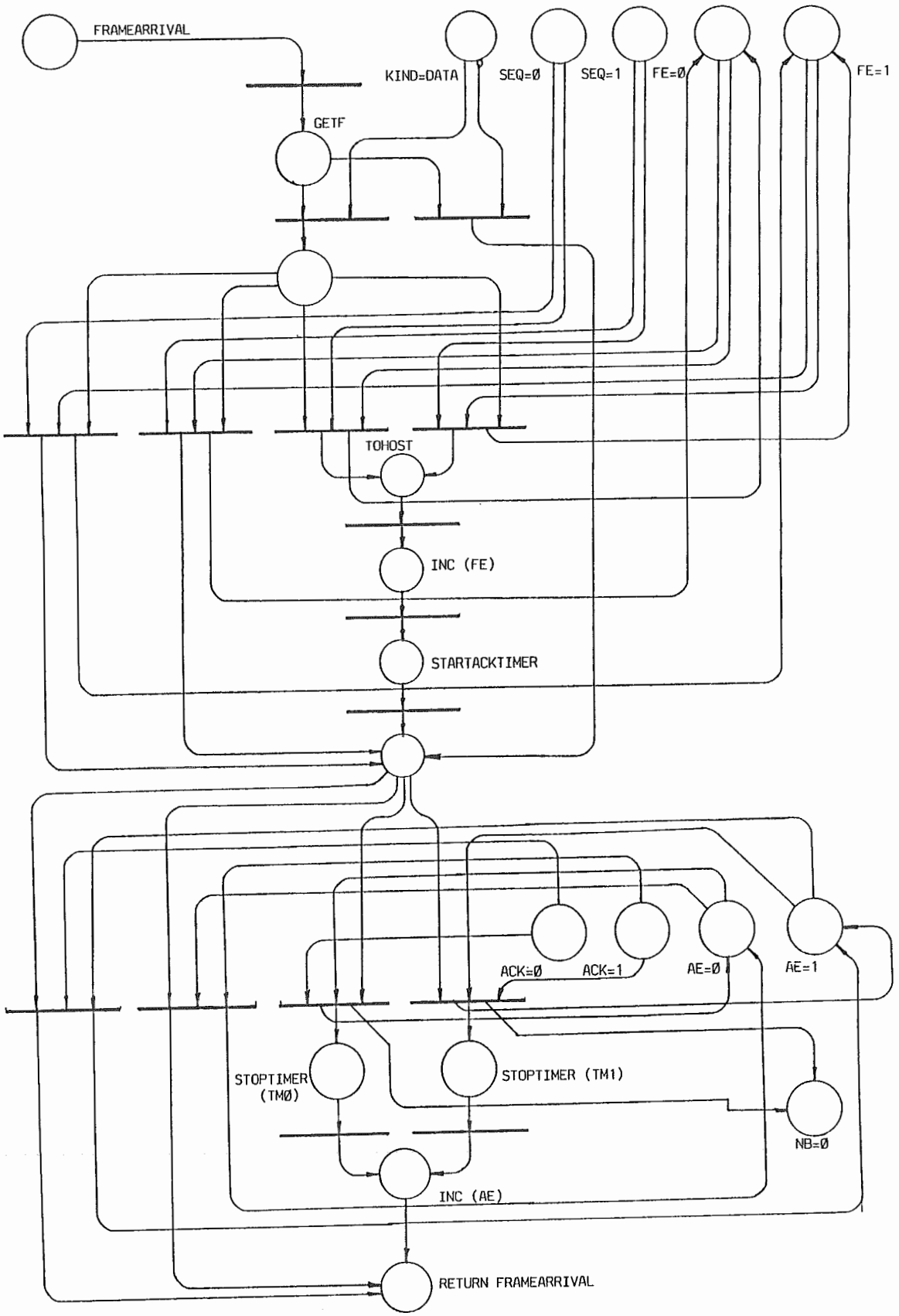


Figura A-3 - Tratamento do Evento FRAMEARRIVAL

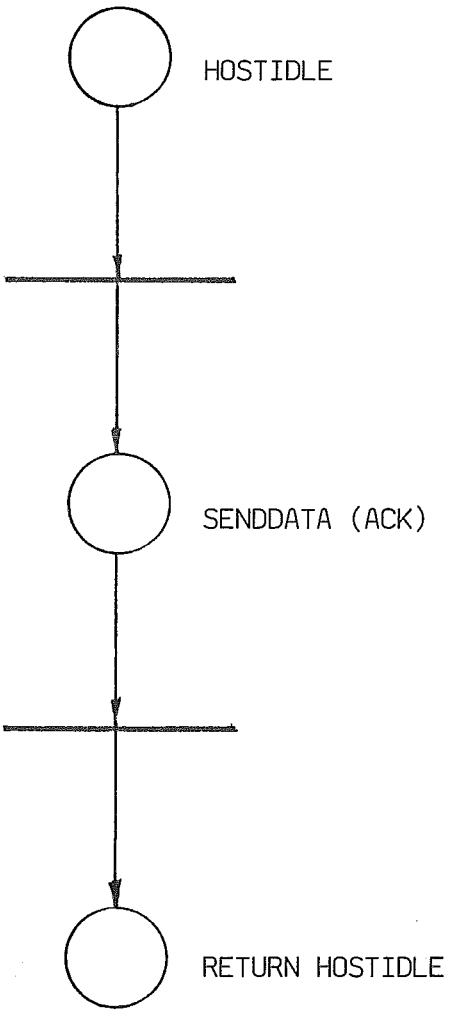


Figura A-4 - Tratamento do Evento HOSTIDLE

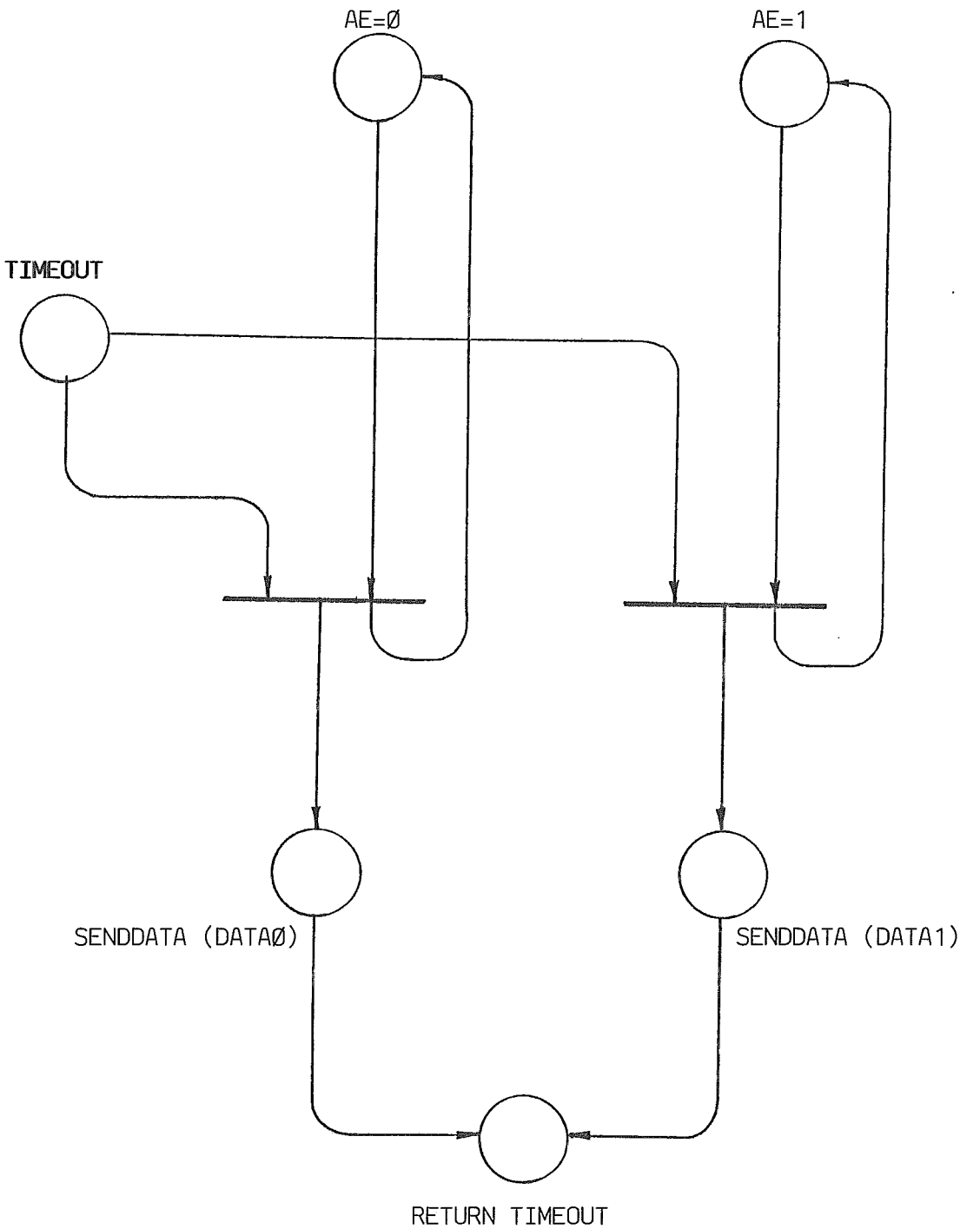


Figura A-5 - Tratamento do Evento TIMEOUT

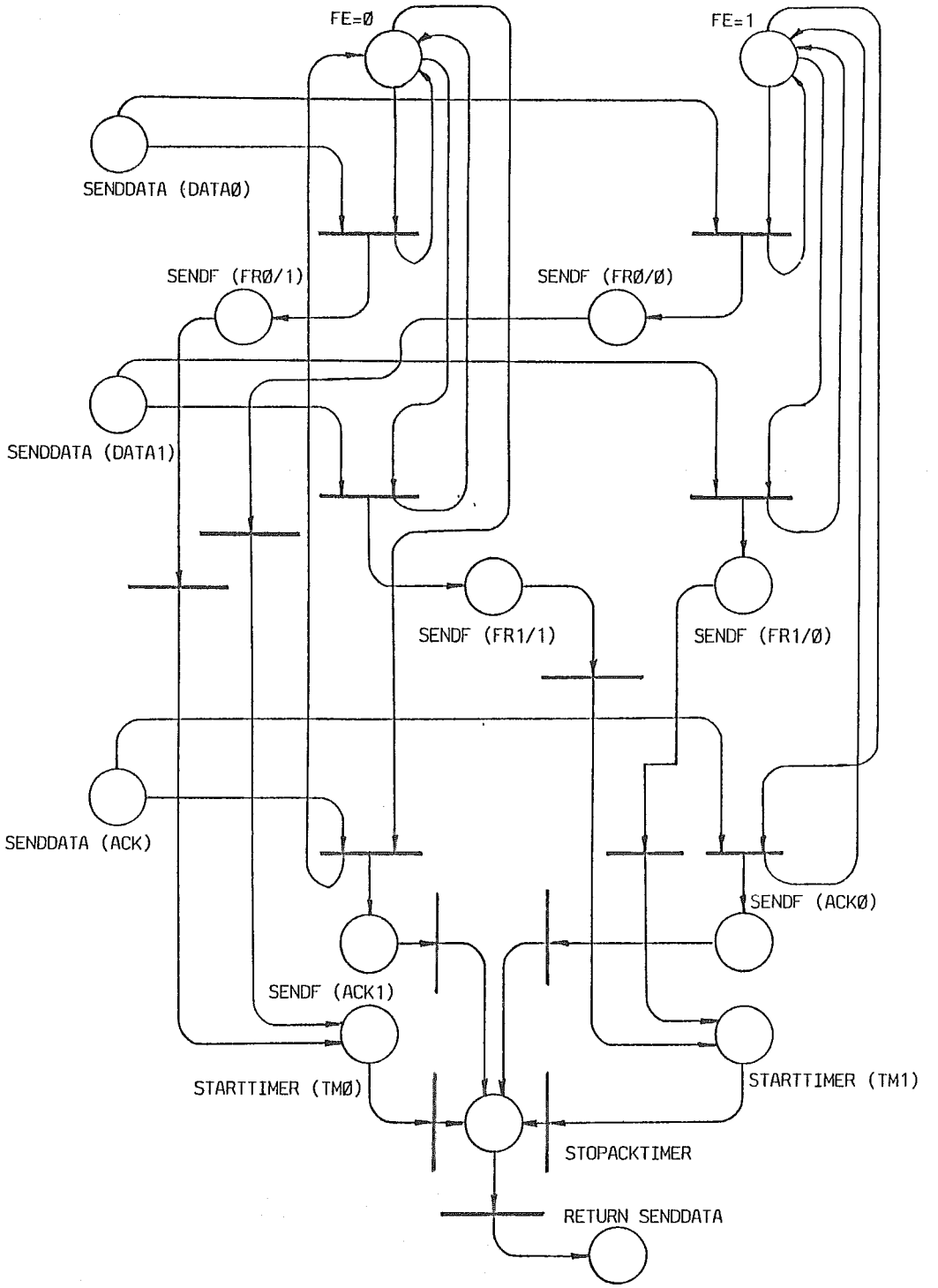


Figura A-6 - Rotina SENDDATA

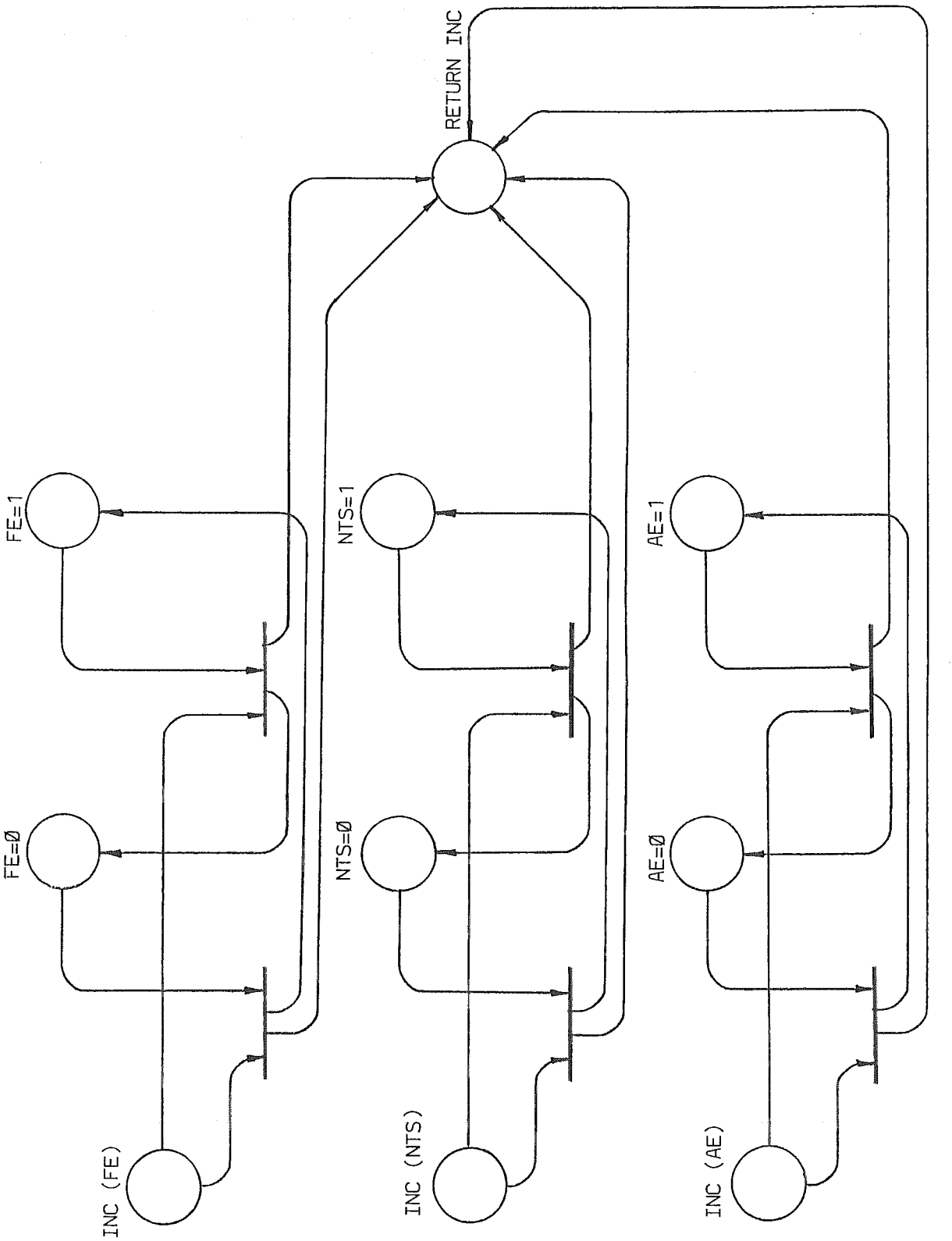


Figura A-7 - Rotina INC

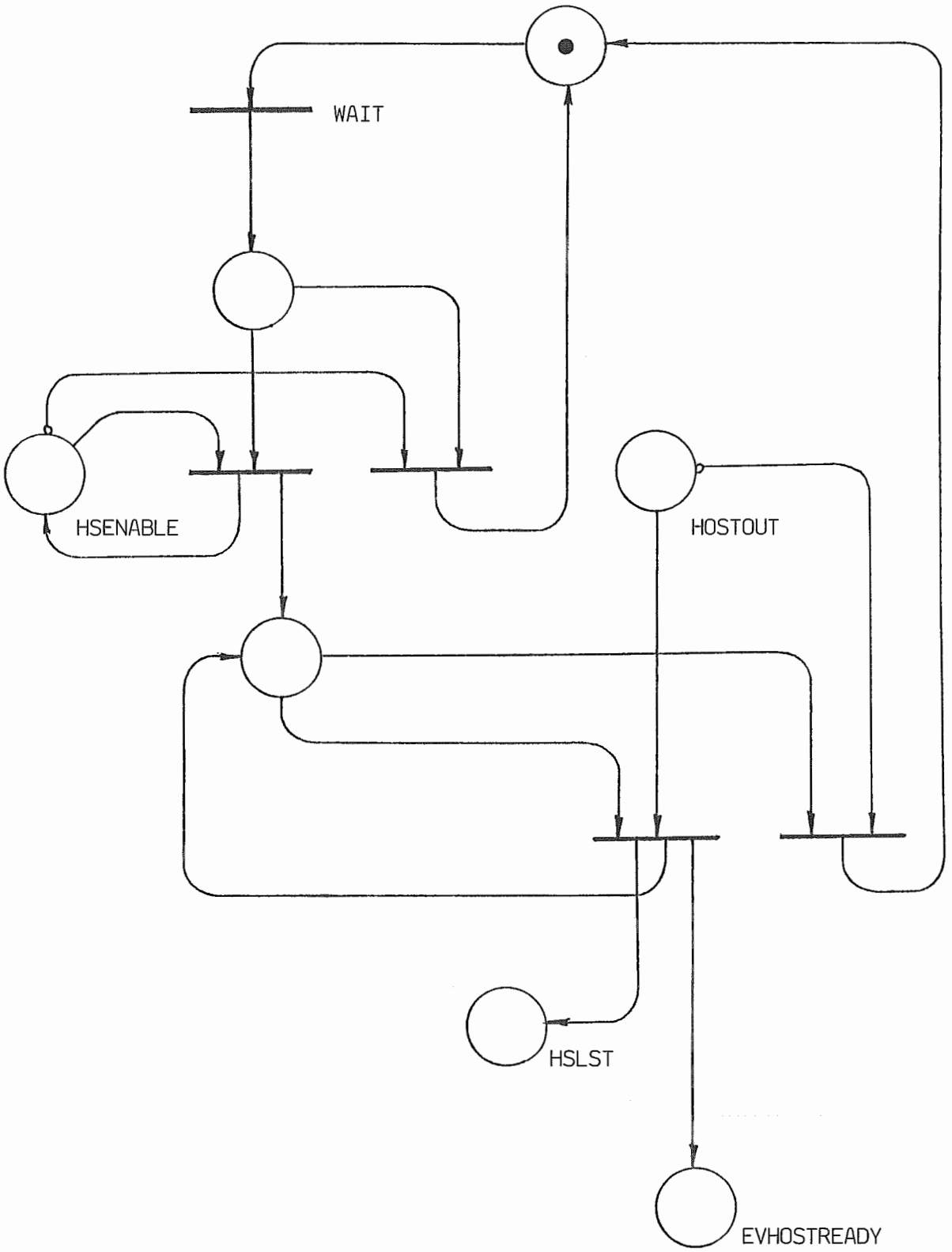


Figura A-8 - Processo HSTASK

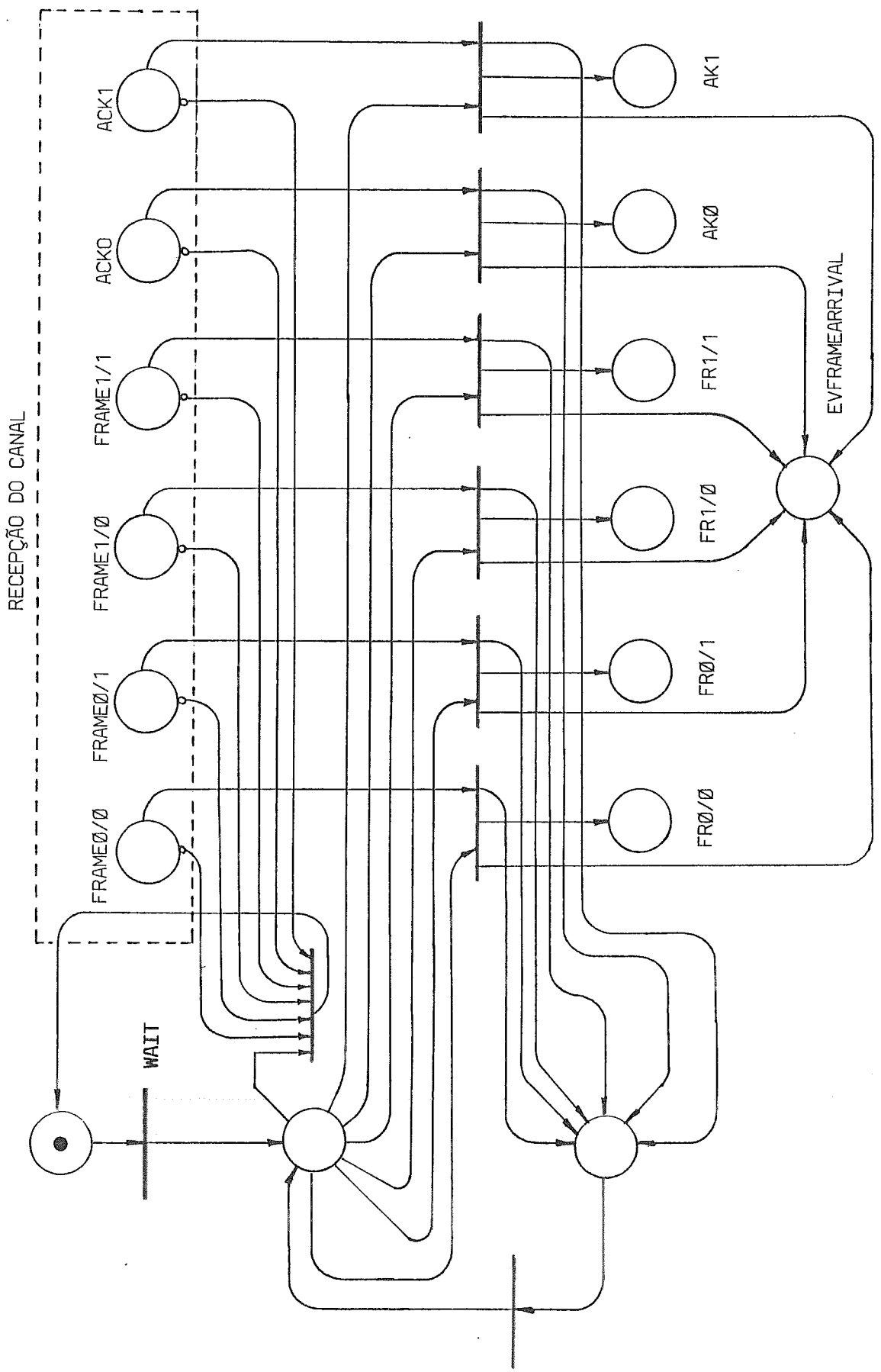


Figura A-9 - Processo NITASK

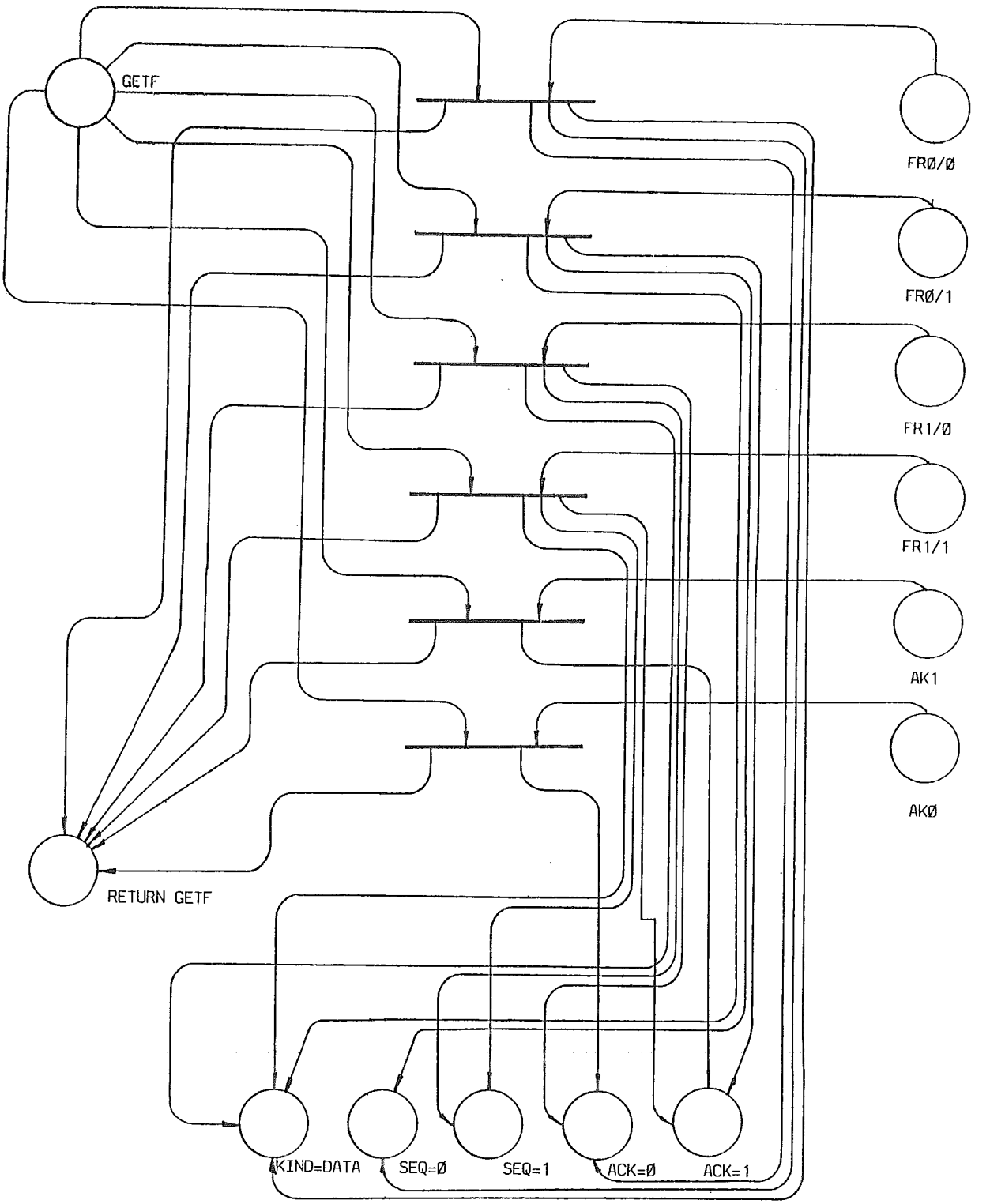
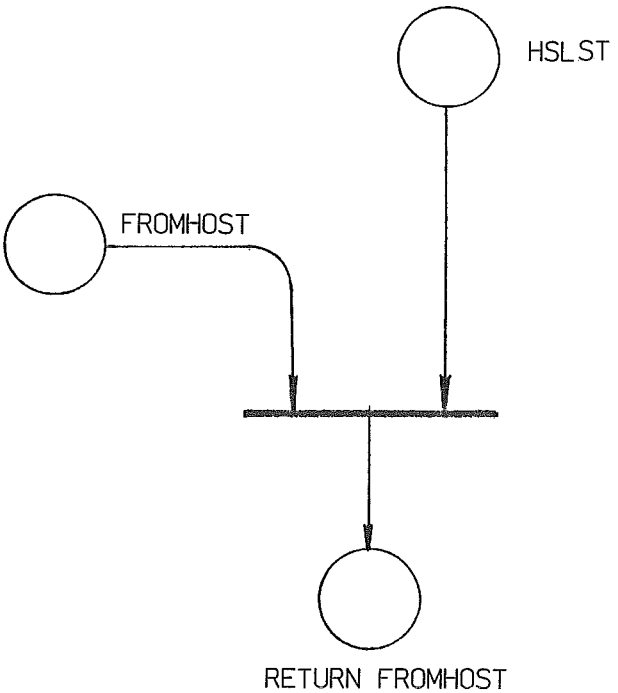


Figura A-10 - Rotina GETF



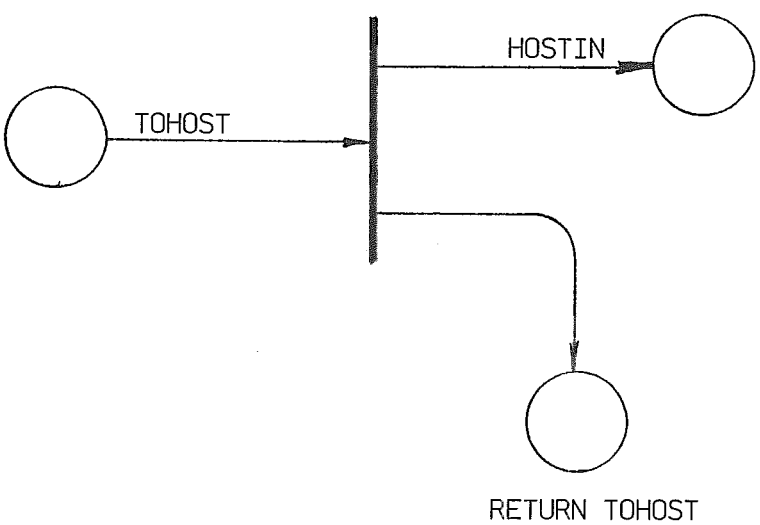


Figura A-11 - Rotinas FROMHOST e TOHOST

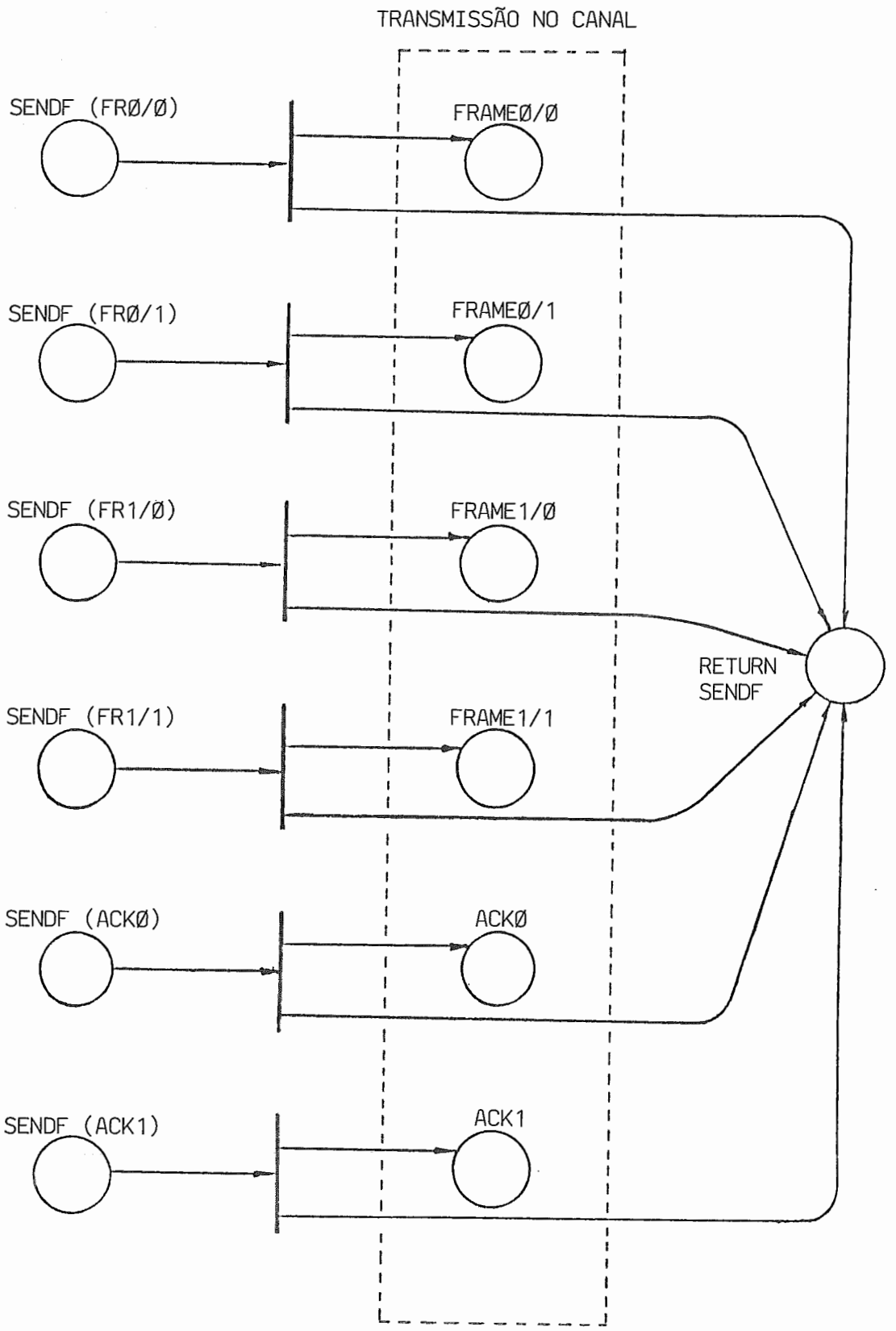


Figura A-12 - Rotina SENDF

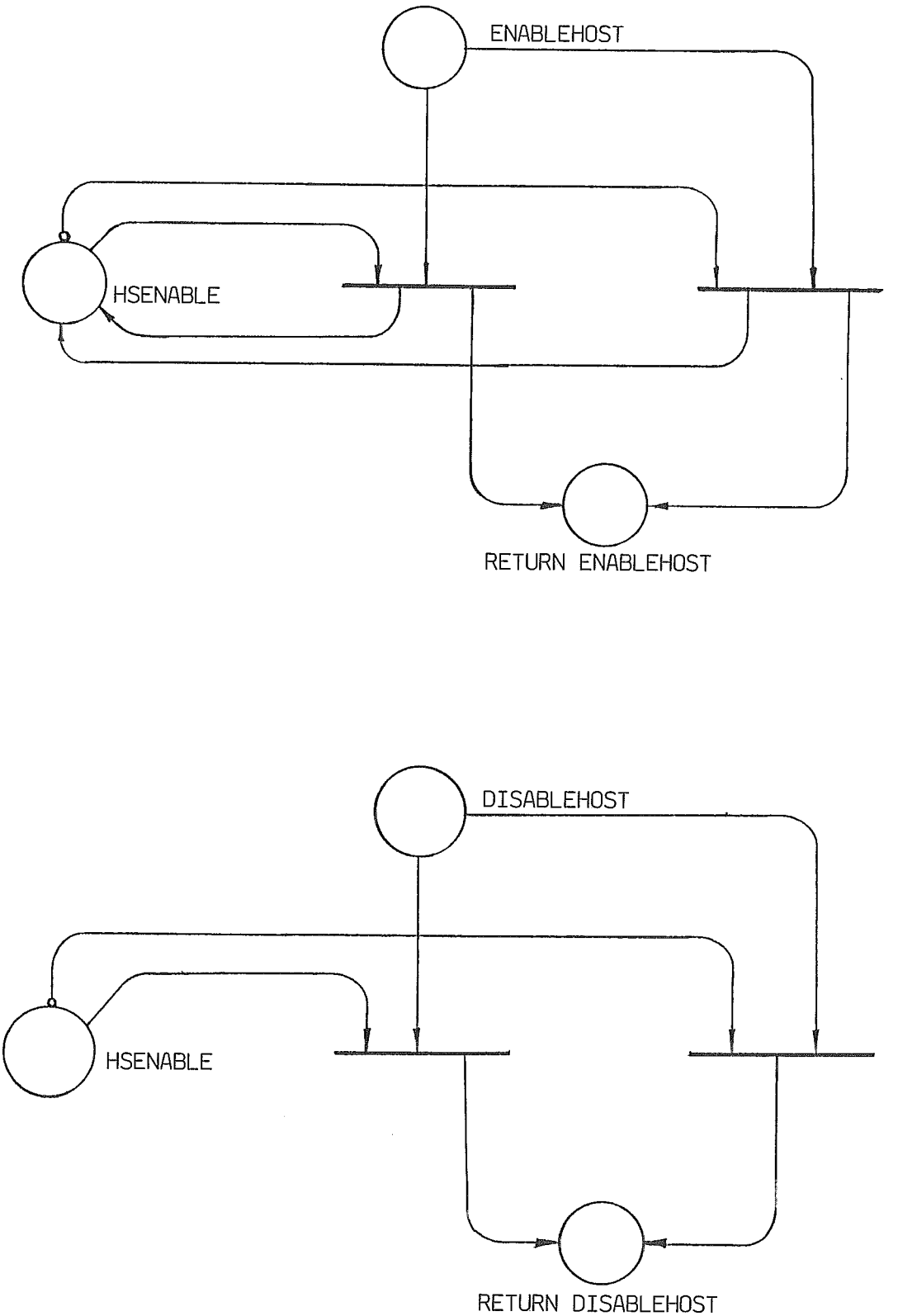


Figura A-13 - Rotinas ENABLEHOST e DISABLEHOST

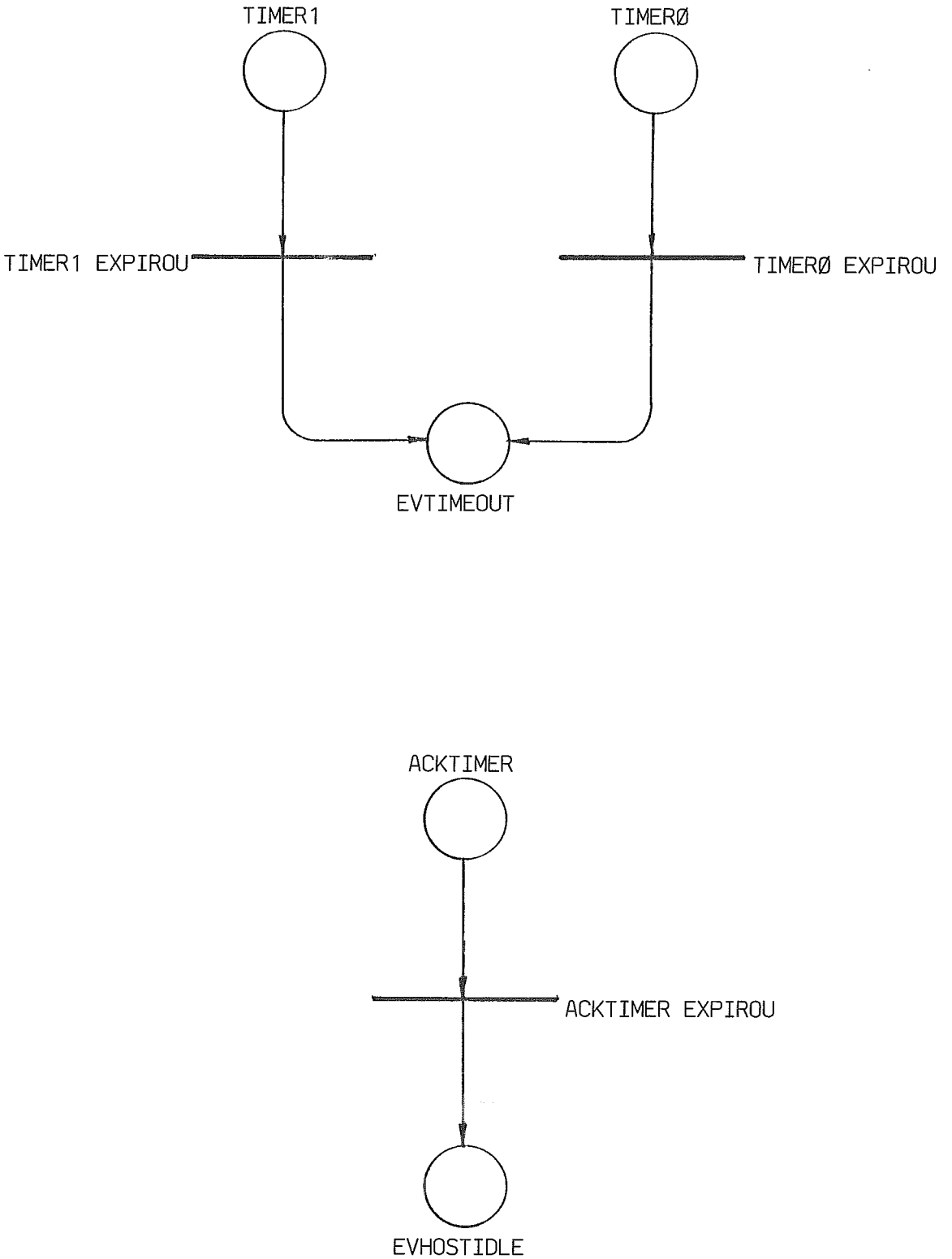


Figura A-14 - Processo TIMERTASK

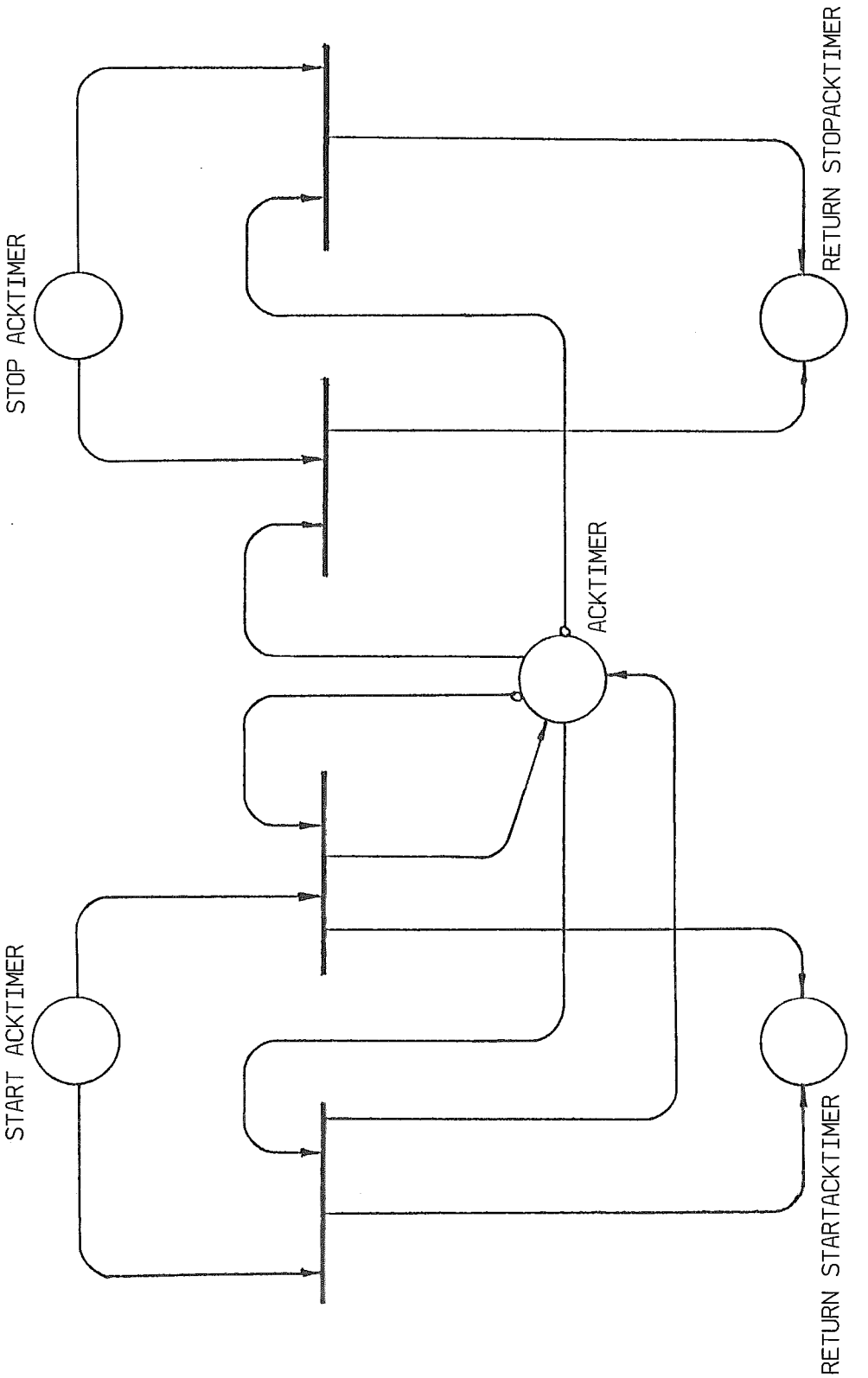


Figura A-15 - Rotinas STARTACKTIMER e STOPACKTIMER

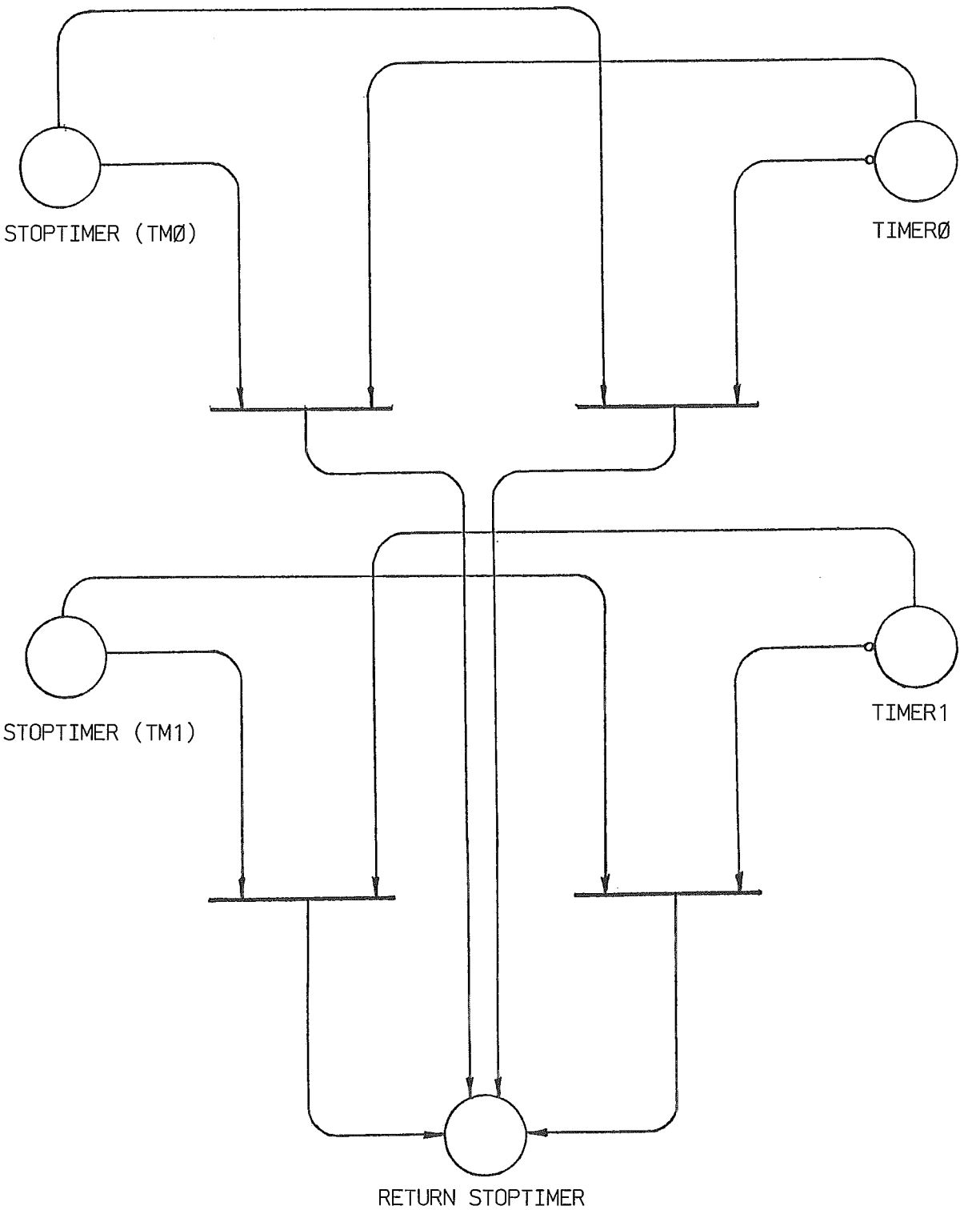


Figura A-16 - Rotina STOPTIMER

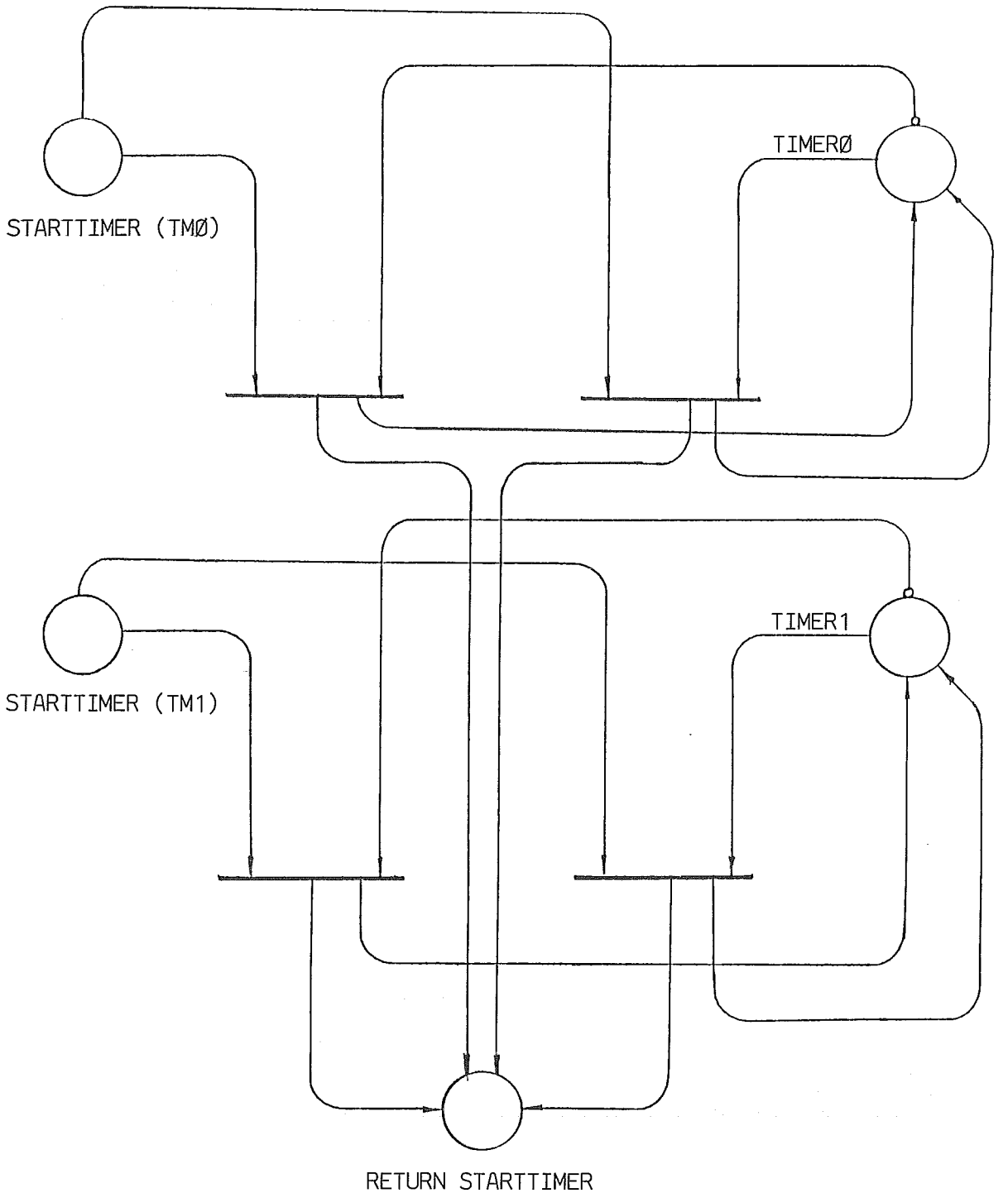


Figura A-17 - Rotina STARTTIMER